

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problems Mailbox.**

THIS PAGE BLANK (USPTO)

Specification of the Bluetooth System

Wireless connections made easy

Core

Bluetooth™

v2.0 B
December 1st 1999

AttributeIDList:**Size: Varies**

Value	Parameter Description
Data Element Sequence	The AttributeIDList is a data element sequence where each element in the list is either an attribute ID or a range of attribute IDs. Each attribute ID is encoded as a 16-bit unsigned integer data element. Each attribute ID range is encoded as a 32-bit unsigned integer data element, where the high order 16 bits are interpreted as the beginning attribute ID of the range and the low order 16 bits are interpreted as the ending attribute ID of the range. The attribute IDs contained in the AttributeIDList must be listed in ascending order without duplication of any attribute ID values. Note that all attributes may be requested by specifying a range of 0x0000-0xFFFF.

ContinuationState:**Size: 1 to 17 Bytes**

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation state information that were returned in a previous response from the server. N is required to be less than or equal to 16. If no continuation state is to be provided in the request, N is set to 0.

4.6.2 SDP_ServiceAttributeResponse PDU

PDU Type	PDU ID	Parameters
SDP_ServiceAttributeResponse	0x05	AttributeListByteCount, AttributeList, ContinuationState

Description:

The SDP server generates an SDP_ServiceAttributeResponse upon receipt of a valid SDP_ServiceAttributeRequest. The response contains a list of attributes (both attribute ID and attribute value) from the requested service record.

PDU Parameters:**AttributeListByteCount:****Size: 2 Bytes**

Value	Parameter Description
N	The AttributeListByteCount contains a count of the number of bytes in the AttributeList parameter. N must never be larger than the MaximumAttributeByteCount value specified in the SDP_ServiceAttributeRequest. Range: 0x0002-0xFFFF

Service Discovery Protocol

Bluetooth.

*AttributeList:**Size: AttributeListByteCount*

Value	Parameter Description
Data Element Sequence	The AttributeList is a data element sequence containing attribute IDs and attribute values. The first element in the sequence contains the attribute ID of the first attribute to be returned. The second element in the sequence contains the corresponding attribute value. Successive pairs of elements in the list contain additional attribute ID and value pairs. Only attributes that have non-null values within the service record and whose attribute IDs were specified in the SDP_ServiceAttributeRequest are contained in the AttributeList. Neither an attribute ID nor an attribute value is placed in the AttributeList for attributes in the service record that have no value. The attributes are listed in ascending order of attribute ID value.

*ContinuationState:**Size: 1 to 17 Bytes*

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation information. If the current response is complete, this parameter consists of a single byte with the value 0. If a partial response is given, the ContinuationState parameter may be supplied in a subsequent request to retrieve the remainder of the response.

4.7 SERVICESEARCHATTRIBUTE TRANSACTION

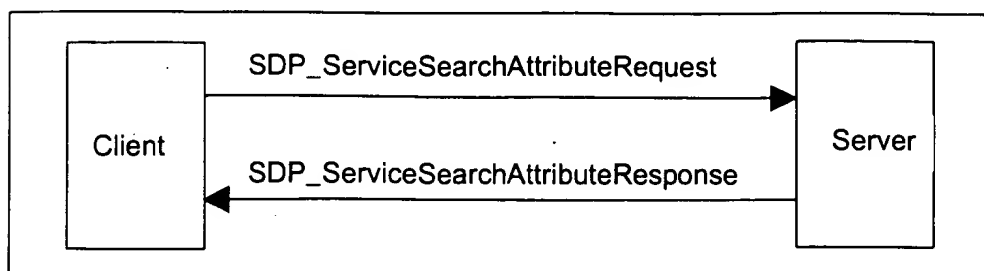


Figure 4.6:

4.7.1 SDP_ServiceSearchAttributeRequest PDU

PDU Type	PDU ID	Parameters
SDP_ServiceSearchAttributeRequest	0x06	ServiceSearchPattern, MaximumAttributeByteCount, AttributeIDList, ContinuationState

Description:

The SDP_ServiceSearchAttributeRequest transaction combines the capabilities of the SDP_ServiceSearchRequest and the SDP_ServiceAttributeRequest into a single request. As parameters, it contains both a service search pattern and a list of attributes to be retrieved from service records that match the service search pattern. The SDP_ServiceSearchAttributeRequest and its response are more complex and may require more bytes than separate SDP_ServiceSearch and SDP_ServiceAttribute transactions. However, using SDP_ServiceSearchAttributeRequest may reduce the total number of SDP transactions, particularly when retrieving multiple service records.

Note that the service record handle for each service record is contained in the ServiceRecordHandle attribute of that service and may be requested along with other attributes.

PDU Parameters:

ServiceSearchPattern:

Size: Varies

Value	Parameter Description
Data Element Sequence	The ServiceSearchPattern is a data element sequence where each element in the sequence is a UUID. The sequence must contain at least one UUID. The maximum number of UUIDs in the sequence is 12*. The list of UUIDs constitutes a service search pattern.

*. The value of 12 has been selected as a compromise between the scope of a service search and the size of a search request PDU. It is not expected that more than 12 UUIDs will be useful in a service search pattern.

MaximumAttributeByteCount:

Size: 2 Bytes

Value	Parameter Description
N	MaximumAttributeByteCount specifies the maximum number of bytes of attribute data to be returned in the response(s) to this request. The SDP server should not return more than N bytes of attribute data in the response(s). If the requested attributes require more than N bytes, the SDP server determines how to truncate the list. Range: 0x0009-0xFFFF

AttributeIDList:

Size: Varies

Value	Parameter Description
Data Element Sequence	The AttributeIDList is a data element sequence where each element in the list is either an attribute ID or a range of attribute IDs. Each attribute ID is encoded as a 16-bit unsigned integer data element. Each attribute ID range is encoded as a 32-bit unsigned integer data element, where the high order 16 bits are interpreted as the beginning attribute ID of the range and the low order 16 bits are interpreted as the ending attribute ID of the range. The attribute IDs contained in the AttributeIDList must be listed in ascending order without duplication of any attribute ID values. Note that all attributes may be requested by specifying a range of 0x0000-0xFFFF.

ContinuationState:

Size: 1 to 17 Bytes

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation state information that were returned in a previous response from the server. N is required to be less than or equal to 16. If no continuation state is to be provided in the request, N is set to 0.

4.7.2 SDP_S rvc SearchAttributeResponse PDU

PDU Type	PDU ID	Parameters
SDP_ServiceSearchAttributeResponse	0x07	AttributeListsByteCount, AttributeLists, ContinuationState

Description:

The SDP server generates an SDP_ServiceSearchAttributeResponse upon receipt of a valid SDP_ServiceSearchAttributeRequest. The response contains a list of attributes (both attribute ID and attribute value) from the service records that match the requested service search pattern.

PDU Parameters:*AttributeListsByteCount:**Size: 2 Bytes*

Value	Parameter Description
N	The AttributeListsByteCount contains a count of the number of bytes in the AttributeLists parameter. N must never be larger than the MaximumAttributeByteCount value specified in the SDP_ServiceSearchAttributeRequest. Range: 0x0002-0xFFFF

*AttributeLists:**Size: Varies*

Value	Parameter Description
Data Element Sequence	The AttributeLists is a data element sequence where each element in turn is a data element sequence representing an attribute list. Each attribute list contains attribute IDs and attribute values from one service record. The first element in each attribute list contains the attribute ID of the first attribute to be returned for that service record. The second element in each attribute list contains the corresponding attribute value. Successive pairs of elements in each attribute list contain additional attribute ID and value pairs. Only attributes that have non-null values within the service record and whose attribute IDs were specified in the SDP_ServiceSearchAttributeRequest are contained in the AttributeLists. Neither an attribute ID nor attribute value is placed in AttributeLists for attributes in the service record that have no value. Within each attribute list, the attributes are listed in ascending order of attribute ID value.

ContinuationState:

Size: 1 to 17 Bytes

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation information. If the current response is complete, this parameter consists of a single byte with the value 0. If a partial response is given, the ContinuationState parameter may be supplied in a subsequent request to retrieve the remainder of the response.

5 SERVICE ATTRIBUTE DEFINITIONS

The service classes and attributes contained in this document are necessarily a partial list of the service classes and attributes supported by SDP. Only service classes that directly support the SDP server are included in this document. Additional service classes will be defined in other documents and possibly in future revisions of this document. Also, it is expected that additional attributes will be discovered that are applicable to a broad set of services; these may be added to the list of Universal attributes in future revisions of this document.

5.1 UNIVERSAL ATTRIBUTE DEFINITIONS

Universal attributes are those service attributes whose definitions are common to all service records. Note that this does not mean that every service record must contain values for all of these service attributes. However, if a service record has a service attribute with an attribute ID allocated to a universal attribute, the attribute value must conform to the universal attribute's definition.

Only two attributes are required to exist in every service record instance. They are the ServiceRecordHandle (attribute ID 0x0000) and the ServiceClassIDList (attribute ID 0x0001). All other service attributes are optional within a service record.

5.1.1 ServiceRecordHandle Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceRecordHandle	0x0000	32-bit unsigned integer

Description:

A service record handle is a 32-bit number that uniquely identifies each service record within an SDP server. It is important to note that, in general, each handle is unique only within each SDP server. If SDP server S1 and SDP server S2 both contain identical service records (representing the same service), the service record handles used to reference these identical service records are completely independent. The handle used to reference the service on S1 will, in general, be meaningless if presented to S2.

5.1.2 ServiceClassIDList Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceClassIDList	0x0001	Data Element Sequence

Description:

The ServiceClassIDList attribute consists of a data element sequence in which each data element is a UUID representing the service classes that a given service record conforms to. The UUIDs are listed in order from the most specific class to the most general class. The ServiceClassIDList must contain at least one service class UUID.

5.1.3 ServiceRecordState Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceRecordState	0x0002	32-bit unsigned integer

Description:

The ServiceRecordState is a 32-bit integer that is used to facilitate caching of ServiceAttributes. If this attribute is contained in a service record, its value is guaranteed to change when any other attribute value is added to, deleted from or changed within the service record. This permits a client to check the value of this single attribute. If its value has not changed since it was last checked, the client knows that no other attribute values within the service record have changed.

5.1.4 ServiceID Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceID	0x0003	UUID

Description:

The ServiceID is a UUID that universally and uniquely identifies the service instance described by the service record. This service attribute is particularly useful if the same service is described by service records in more than one SDP server.

5.1.5 Protoc ID scriptorList Attribute

Attribute Name	Attribute ID	Attribute Value Type
ProtocolDescriptorList	0x0004	Data Element Sequence or Data Element Alternative

Description:

The ProtocolDescriptorList attribute describes one or more protocol stacks that may be used to gain access to the service described by the service record.

If the ProtocolDescriptorList describes a single stack, it takes the form of a data element sequence in which each element of the sequence is a protocol descriptor. Each protocol descriptor is, in turn, a data element sequence whose first element is a UUID identifying the protocol and whose successive elements are protocol-specific parameters. Potential protocol-specific parameters are a protocol version number and a connection-port number. The protocol descriptors are listed in order from the lowest layer protocol to the highest layer protocol used to gain access to the service.

If it is possible for more than one kind of protocol stack to be used to gain access to the service, the ProtocolDescriptorList takes the form of a data element alternative where each member is a data element sequence as described in the previous paragraph.

Protocol Descriptors

A protocol descriptor identifies a communications protocol and provides protocol-specific parameters. A protocol descriptor is represented as a data element sequence. The first data element in the sequence must be the UUID that identifies the protocol. Additional data elements optionally provide protocol-specific information, such as the L2CAP protocol/service multiplexer (PSM) and the RFCOMM server channel number (CN) shown below.

ProtocolDescriptorList Examples

These examples are intended to be illustrative. The parameter formats for each protocol are not defined within this specification.

In the first two examples, it is assumed that a single RFCOMM instance exists on top of the L2CAP layer. In this case, the L2CAP protocol specific information (PSM) points to the single instance of RFCOMM. In the last example, two different and independent RFCOMM instances are available on top of the L2CAP layer. In this case, the L2CAP protocol specific information (PSM) points to a distinct identifier that distinguishes each of the RFCOMM instances. According to the L2CAP specification, this identifier takes values in the range 0x1000-0xFFFF.

IrDA-like printer

```
(( L2CAP, PSM=RFCOMM ), ( RFCOMM, CN=1 ), ( PostscriptStream ) )
```

IP Network Printing

```
(( L2CAP, PSM=RFCOMM ), ( RFCOMM, CN=2 ), ( PPP ), ( IP ), ( TCP ),  
( IPP ) )
```

Synchronization Protocol Descriptor Example

```
(( L2CAP, PSM=0x1001 ), ( RFCOMM, CN=1 ), ( Obex ), ( vCal ) )
```

```
(( L2CAP, PSM=0x1002 ), ( RFCOMM, CN=1 ), ( Obex ),  
( otherSynchronisationApplication ) )
```

5.1.6 BrowseGroupList Attribute

Attribute Name	Attribute ID	Attribute Value Type
BrowseGroupList	0x0005	Data Element Sequence

Description:

The BrowseGroupList attribute consists of a data element sequence in which each element is a UUID that represents a browse group to which the service record belongs. The top-level browse group ID, called PublicBrowseRoot and representing the root of the browsing hierarchy, has the value 00001002-0000-1000-7007-00805F9B34FB (UUID16: 0x1002) from the Bluetooth Assigned Numbers document.

5.1.7 LanguageBaseAttributeIDList Attribute

Attribute Name	Attribute ID	Attribute Value Type
LanguageBaseAttributeIDList	0x0006	Data Element Sequence

Description:

In order to support human-readable attributes for multiple natural languages in a single service record, a base attribute ID is assigned for each of the natural languages used in a service record. The human-readable universal attributes are then defined with an attribute ID offset from each of these base values, rather than with an absolute attribute ID.

The LanguageBaseAttributeIDList attribute is a list in which each member contains a language identifier, a character encoding identifier, and a base attribute

Bluetooth.

ID for each of the natural languages used in the service record. The LanguageBaseAttributeIDList attribute consists of a data element sequence in which each element is a 16-bit unsigned integer. The elements are grouped as triplets (threes).

The first element of each triplet contains an identifier representing the natural language. The language is encoded according to ISO 639:1988 (E/F): "Code for the representation of names of languages".

The second element of each triplet contains an identifier that specifies a character encoding used for the language. Values for character encoding can be found in IANA's database², and have the values that are referred to as MIBEnum values. The recommended character encoding is UTF-8.

The third element of each triplet contains an attribute ID that serves as the base attribute ID for the natural language in the service record. Different service records within a server may use different base attribute ID values for the same language.

To facilitate the retrieval of human-readable universal attributes in a principal language, the base attribute ID value for the primary language supported by a service record must be 0x0100. Also, if a LanguageBaseAttributeIDList attribute is contained in a service record, the base attribute ID value contained in its first element must be 0x0100.

5.1.8 ServiceInfoTimeToLive Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceInfoTimeToLive	0x0007	32-bit unsigned integer

Description:

The ServiceTimeToLive attribute is a 32-bit integer that contains the number of seconds for which the information in a service record is expected to remain valid and unchanged. This time interval is measured from the time that the attribute value is retrieved from the SDP server. This value does not imply a guarantee that the service record will remain available or unchanged. It is simply a hint that a client may use to determine a suitable polling interval to re-validate the service record contents.

2. See <http://www.isi.edu/in-notes/iana/assignments/character-sets>

5.1.9 ServiceAvailability Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceAvailability	0x0008	8-bit unsigned integer

Description:

The ServiceAvailability attribute is an 8-bit unsigned integer that represents the relative ability of the service to accept additional clients. A value of 0xFF indicates that the service is not currently in use and is thus fully available, while a value of 0x00 means that the service is not accepting new clients. For services that support multiple simultaneous clients, intermediate values indicate the relative availability of the service on a linear scale.

For example, a service that can accept up to 3 clients should provide ServiceAvailability values of 0xFF, 0xAA, 0x55, and 0x00 when 0, 1, 2, and 3 clients, respectively, are utilising the service. The value 0xAA is approximately $(2/3) * 0xFF$ and represents 2/3 availability, while the value 0x55 is approximately $(1/3) * 0xFF$ and represents 1/3 availability. Note that the availability value may be approximated as

$$(1 - (\text{current_number_of_clients} / \text{maximum_number_of_clients})) * 0xFF$$

When the maximum number of clients is large, this formula must be modified to ensure that ServiceAvailability values of 0x00 and 0xFF are reserved for their defined meanings of unavailability and full availability, respectively.

Note that the maximum number of clients a service can support may vary according to the resources utilised by the service's current clients.

A non-zero value for ServiceAvailability does not guarantee that the service will be available for use. It should be treated as a hint or an approximation of availability status.

5.1.10 BluetoothProfileDescriptorList Attribute

Attribute Name	Attribute ID	Attribute Value Type
BluetoothProfileDescriptorList	0x0009	Data Element Sequence

Description:

The BluetoothProfileDescriptorList attribute consists of a data element sequence in which each element is a profile descriptor that contains information about a Bluetooth profile to which the service represented by this service record conforms. Each profile descriptor is a data element sequence whose

Bluetooth.

first element is the UUID assigned to the profile and whose second element is a 16-bit profile version number.

Each version of a profile is assigned a 16-bit unsigned integer profile version number, which consists of two 8-bit fields. The higher-order 8 bits contain the major version number field and the lower-order 8 bits contain the minor version number field. The initial version of each profile has a major version of 1 and a minor version of 0. When upward compatible changes are made to the profile, the minor version number will be incremented. If incompatible changes are made to the profile, the major version number will be incremented.

5.1.11 DocumentationURL Attribute

Attribute Name	Attribute ID	Attribute Value Type
DocumentationURL	0x000A	URL

Description:

This attribute is a URL which points to documentation on the service described by a service record.

5.1.12 ClientExecutableURL Attribute

Attribute Name	Attribute ID	Attribute Value Type
ClientExecutableURL	0x000B	URL

Description:

This attribute contains a URL that refers to the location of an application that may be used to utilize the service described by the service record. Since different operating environments require different executable formats, a mechanism has been defined to allow this single attribute to be used to locate an executable that is appropriate for the client device's operating environment. In the attribute value URL, the first byte with the value 0x2A (ASCII character '*') is to be replaced by the client application with a string representing the desired operating environment before the URL is to be used.

The list of standardized strings representing operating environments is contained in the Bluetooth Assigned Numbers document.

For example, assume that the value of the ClientExecutableURL attribute is `http://my.fake/public/*/client.exe`. On a device capable of executing SH3 WindowsCE files, this URL would be changed to `http://my.fake/public/sh3-microsoft-wince/client.exe`. On a device capable of executing Windows 98 binaries, this URL would be changed to `http://my.fake/public/i86-microsoft-win98/client.exe`.

5.1.13 IconURL Attribute

Attribute Name	Attribute ID	Attribute Value Type
IconURL	0x000C	URL

Description:

This attribute contains a URL that refers to the location of an icon that may be used to represent the service described by the service record. Since different hardware devices require different icon formats, a mechanism has been defined to allow this single attribute to be used to locate an icon that is appropriate for the client device. In the attribute value URL, the first byte with the value 0x2A (ASCII character '*') is to be replaced by the client application with a string representing the desired icon format before the URL is to be used.

The list of standardized strings representing icon formats is contained in the Bluetooth Assigned Numbers document.

For example, assume that the value of the IconURL attribute is `http://my.fake/public/icons/*`. On a device that prefers 24 x 24 icons with 256 colors, this URL would be changed to `http://my.fake/public/icons/24x24x8.png`. On a device that prefers 10 x 10 monochrome icons, this URL would be changed to `http://my.fake/public/icons/10x10x1.png`.

5.1.14 ServiceName Attribute

Attribute Name	Attribute ID Offset	Attribute Value Type
ServiceName	0x0000	String

Description:

The ServiceName attribute is a string containing the name of the service represented by a service record. It should be brief and suitable for display with an icon representing the service. The offset 0x0000 must be added to the attribute ID base (contained in the LanguageBaseAttributeIDList attribute) in order to compute the attribute ID for this attribute.

5.1.15 ServiceDescription Attribute

Attribute Name	Attribute ID Offset	Attribute Value Type
ServiceDescription	0x0001	String

Description:

This attribute is a string containing a brief description of the service. It should be less than 200 characters in length. The offset 0x0001 must be added to the attribute ID base (contained in the LanguageBaseAttributeIDList attribute) in order to compute the attribute ID for this attribute.

5.1.16 ProviderName Attribute

Attribute Name	Attribute ID Offset	Attribute Value Type
ProviderName	0x0002	String

Description:

This attribute is a string containing the name of the person or organization providing the service. The offset 0x0002 must be added to the attribute ID base (contained in the LanguageBaseAttributeIDList attribute) in order to compute the attribute ID for this attribute.

5.1.17 Reserved Universal Attribute IDs

Attribute IDs in the range of 0x000D-0x01FF are reserved.

5.2 SERVICEDISCOVERYSERVER SERVICE CLASS ATTRIBUTE DEFINITIONS

This service class describes service records that contain attributes of service discovery server itself. The attributes listed in this section are only valid if the ServiceClassIDList attribute contains the ServiceDiscoveryServerServiceClassID. Note that all of the universal attributes may be included in service records of the ServiceDiscoveryServer class.

5.2.1 ServiceRecordHandle Attribute

Described in the universal attribute definition for ServiceRecordHandle.

Value

A 32-bit integer with the value 0x00000000.

5.2.2 ServiceClassIDList Attribute

Described in the universal attribute definition for ServiceClassIDList.

Value

A UUID representing the ServiceDiscoveryServerServiceClassID.

5.2.3 VersionNumberList Attribute

Attribute Name	Attribute ID	Attribute Value Type
VersionNumberList	0x0200	Data Element Sequence

Description:

The VersionNumberList is a data element sequence in which each element of the sequence is a version number supported by the SDP server.

A version number is a 16-bit unsigned integer consisting of two fields. The higher-order 8 bits contain the major version number field and the low-order 8 bits contain the minor version number field. The initial version of SDP has a major version of 1 and a minor version of 0. When upward compatible changes are made to the protocol, the minor version number will be incremented. If incompatible changes are made to SDP, the major version number will be incremented. This guarantees that if a client and a server support a common major version number, they can communicate if each uses only features of the specification with a minor version number that is supported by both client and server.

5.2.4 ServiceDatabaseState Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceDatabaseState	0x0201	32-bit unsigned integer

Description:

The ServiceDatabaseState is a 32-bit integer that is used to facilitate caching of service records. If this attribute exists, its value is guaranteed to change when any of the other service records are added to or deleted from the server's database. If this value has not changed since the last time a client queried its value, the client knows that a) none of the other service records maintained by the SDP server have been added or deleted; and b) any service record handles acquired from the server are still valid. A client should query this attribute's value when a connection to the server is established, prior to using any service record handles acquired during a previous connection.

Note that the ServiceDatabaseState attribute does not change when existing service records are modified, including the addition, removal, or modification of service attributes. A service record's ServiceRecordState attribute indicates when that service record is modified.

5.2.5 Reserved Attribute IDs

Attribute IDs in the range of 0x0202-0x02FF are reserved.

5.3 BROWSEGROUPDESCRIPTOR SERVICE CLASS ATTRIBUTE DEFINITIONS

This service class describes the ServiceRecord provided for each Browse-GroupDescriptor service offered on a Bluetooth device. The attributes listed in this section are only valid if the ServiceClassIDList attribute contains the BrowseGroupDescriptorServiceClassID. Note that all of the universal attributes may be included in service records of the BrowseGroupDescriptor class.

5.3.1 ServiceClassIDList Attribute

Described in the universal attribute definition for ServiceClassIDList.

Value

A UUID representing the BrowseGroupDescriptorServiceClassID.

5.3.2 GroupID Attribute

Attribute Name	Attribute ID	Attribute Value Type
GroupID	0x0200	UUID

Description:

This attribute contains a UUID that can be used to locate services that are members of the browse group that this service record describes.

5.3.3 Reserved Attribute IDs

Attribute IDs in the range of 0x0201-0x02FF are reserved.

APPENDIX A – BACKGROUND INFORMATION

A.1. Service Discovery

As computing continues to move to a network-centric model, finding and making use of services that may be available in the network becomes increasingly important. Services can include common ones such as printing, paging, FAX-ing, and so on, as well as various kinds of information access such as teleconferencing, network bridges and access points, eCommerce facilities, and so on — most any kind of service that a server or service provider might offer. In addition to the need for a standard way of discovering available services, there are other considerations: getting access to the services (finding and obtaining the protocols, access methods, “drivers” and other code necessary to utilize the service), controlling access to the services, advertising the services, choosing among competing services, billing for services, and so on. This problem is widely recognized; many companies, standards bodies and consortia are addressing it at various levels in various ways. Service Location Protocol (SLP), JiniTM, and SalutationTM, to name just a few, all address some aspect of service discovery.

A.2. Bluetooth Service Discovery

Bluetooth Service Discovery Protocol (SDP) addresses service discovery specifically for the Bluetooth environment. It is optimized for the highly dynamic nature of Bluetooth communications. SDP focuses primarily on discovering services available from or through Bluetooth devices. SDP does not define methods for accessing services; once services are discovered with SDP, they can be accessed in various ways, depending upon the service. This might include the use of other service discovery and access mechanisms such as those mentioned above; SDP provides a means for other protocols to be used along with SDP in those environments where this can be beneficial. While SDP can coexist with other service discovery protocols, it does not require them. In Bluetooth environments, services can be discovered using SDP and can be accessed using other protocols defined by Bluetooth.

APPENDIX B – EXAMPLE SDP TRANSACTIONS

The following are simple examples of typical SDP transactions. These are meant to be illustrative of SDP flows. The examples do not consider:

- Caching (in a caching system, the SDP client would make use of the ServiceRecordState and ServiceDatabaseState attributes);
- Service availability (if this is of interest, the SDP client should use the ServiceAvailability and/or ServiceTimeToLive attributes);
- SDP versions (the VersionNumberList attribute could be used to determine compatible SDP versions);
- SDP Error Responses (an SDP error response is possible for any SDP request that is in error); and
- Communication connection (the examples assume that an L2CAP connection is established).

The examples are meant to be illustrative of the protocol. The format used is `ObjectName[ObjectSizeInBytes] {SubObjectDefinitions}`, but this is not meant to illustrate an interface. The `ObjectSizeInBytes` is the size of the object in decimal. The `SubObjectDefinitions` (inside of `{}` characters) are components of the immediately enclosing object. Hexadecimal values shown as lower-case letters, such as for transaction IDs and service handles, are variables (the particular value is not important for the illustration, but each such symbol always represents the same value). Comments are included in this manner: `/* comment text */`.

B.1. SDP Example 1 – ServiceSearchRequest

The first example is that of an SDP client searching for a generic printing service. The client does not specify a particular type of printing service. In the example, the SDP server has two available printing services. The transaction illustrates:

1. SDP client to SDP server: `SDP_ServiceSearchRequest`, specifying the `PrinterServiceClassID` (represented as a `DataElement` with a 32-bit UUID value of `ppp . . . ppp`) as the only element of the `ServiceSearchPattern`. The `PrinterServiceClassID` is assumed to be a 32-bit UUID and the data element type for it is illustrated. The `TransactionID` is illustrated as `tttt`.
2. SDP server to SDP client: `SDP_ServiceSearchResponse`, returning handles to two printing services, represented as `qqqqqqqq` for the first printing service and `rrrrrrrr` for the second printing service. The `TransactionID` is the same value as supplied by the SDP client in the corresponding request (`tttt`).

```
/* Sent from SDP Client to SDP server */
SDP_ServiceSearchRequest[15] {
    PDUID[1] {
```

*Service Discovery Protocol***Bluetooth.**

```

    0x02
  }
  TransactionID[2] {
    0xtttt
  }
  ParameterLength[2] {
    0x000A
  }
  ServiceSearchPattern[7] {
    DataElementSequence[7] {
      0b00110 0b101 0x05
      UUID[5] {
        /* PrinterServiceClassID */
        0b00011 0b010 0xppppppppp
      }
    }
  }
  MaximumServiceRecordCount[2] {
    0x0003
  }
  ContinuationState[1] {
    /* no continuation state */
    0x00
  }
}

/* Sent from SDP server to SDP client */
SDP_ServiceSearchResponse[16] {
  PDUID[1] {
    0x03
  }
  TransactionID[2] {
    0xtttt
  }
  ParameterLength[2] {
    0x000D
  }
  TotalServiceRecordCount[2] {
    0x0002
  }
  CurrentServiceRecordCount[2] {
    0x0002
  }
  ServiceRecordHandleList[8] {
    /* print service 1 handle */
    0xqqqqqqqq
    /* print service 2 handle */
    0xxxxrrrrrr
  }
  ContinuationState[1] {
    /* no continuation state */
    0x00
  }
}

```

B.2. SDP Example 2 – ServiceAttributeTransaction

The second example continues the first example. In Example 1, the SDP client obtained handles to two printing services. In Example 2, the client uses one of those service handles to obtain the ProtocolDescriptorList attribute for that printing service. The transaction illustrates:

1. SDP client to SDP server: SDP_ServiceAttributeRequest, presenting the previously obtained service handle (the one denoted as qqqqqqqq) and specifying the ProtocolDescriptorList attribute ID (AttributeID 0x0004) as the only attribute requested (other attributes could be retrieved in the same transaction if desired). The TransactionID is illustrated as uuuu to distinguish it from the TransactionID of Example 1.
2. SDP server to SDP client: SDP_ServiceAttributeResponse, returning the ProtocolDescriptorList for the specified printing service. This protocol stack is assumed to be (L2CAP), (RFCOMM, 2), (PostscriptStream). The ProtocolDescriptorList is a data element sequence in which each element is, in turn, a data element sequence whose first element is a UUID representing the protocol, and whose subsequent elements are protocol-specific parameters. In this example, one such parameter is included for the RFCOMM protocol, an 8-bit value indicating RFCOMM server channel 2. The Transaction ID is the same value as supplied by the SDP client in the corresponding request (uuuu). The Attributes returned are illustrated as a data element sequence where the protocol descriptors are 32-bit UUIDs and the RFCOMM server channel is a data element with an 8-bit value of 2.

```

/* Sent from SDP Client to SDP server */
SDP_ServiceAttributeRequest[17] {
  PDUID[1] {
    0x04
  }
  TransactionID[2] {
    0xuuuu
  }
  ParameterLength[2] {
    0x000C
  }
  ServiceRecordHandle[4] {
    0xqqqqqqqq
  }
  MaximumAttributeByteCount[2] {
    0x0080
  }
  AttributeIDList[5] {
    DataElementSequence[5] {
      0b00110 0b101 0x03
      AttributeID[3] {
        0b00001 0b001 0x0004
      }
    }
  }
}

```

*Service Discovery Protocol***Bluetooth.**

```

ContinuationState[1] {
    /* no continuation state */
    0x00
}
}

/* Sent from SDP server to SDP client */
SDP_ServiceAttributeResponse[36] {
    PDUID[1] {
        0x05
    }
    TransactionID[2] {
        0xuuuu,
    }
    ParameterLength[2] {
        0x0021
    }
    AttributeListByteCount[2] {
        0x001E
    }
    AttributeList[30] {
        DataElementSequence[30] {
            0b00110 0b101 0x1C
            Attribute[28] {
                AttributeID[3] {
                    0b00001 0b001 0x0004
                }
                AttributeValue[25] {
                    /* ProtocolDescriptorList */
                    DataElementSequence[25] {
                        0b00110 0b101 0x17
                        /* L2CAP protocol descriptor */
                        DataElementSequence[7] {
                            0b00110 0b101 0x05
                            UUID[5] {
                                /* L2CAP Protocol UUID */
                                0b00011 0b010 <32-bit L2CAP UUID>
                            }
                        }
                    }
                    /* RFCOMM protocol descriptor */
                    DataElementSequence[9] {
                        0b00110 0b101 0x07
                        UUID[5] {
                            /* RFCOMM Protocol UUID */
                            0b00011 0b010 <32-bit RFCOMM UUID>
                        }
                    }
                    /* parameter for server 2 */
                    Uint8[2] {
                        0b00001 0b000 0x02
                    }
                }
            }
        }
        /* PostscriptStream protocol descriptor */
        DataElementSequence[7] {
            0b00110 0b101 0x05
            UUID[5] {
                /* PostscriptStream Protocol UUID */
                0b00011 0b010 <32-bit PostscriptStream UUID>
            }
        }
    }
}

```

*Service Discovery Protocol***Bluetooth.**

```
    }  
  }  
}  
ContinuationState[1] {  
  /* no continuation state */  
  0x00  
}
```

B.3. SDP Example 3 – ServiceSearchAttributeTransaction

The third example is a form of service browsing, although it is not generic browsing in that it does not make use of SDP browse groups. Instead, an SDP client is searching for available Synchronization services that can be presented to the user for selection. The SDP client does not specify a particular type of synchronization service. In the example, the SDP server has three available synchronization services: an address book synchronization service and a calendar synchronization service (both from the same provider), and a second calendar synchronization service from a different provider. The SDP client is retrieving the same attributes for each of these services; namely, the data formats supported for the synchronization service (vCard, vCal, iCal, etc.) and those attributes that are relevant for presenting information to the user about the services. Also assume that the maximum size of a response is 400 bytes. Since the result is larger than this, the SDP client will repeat the request supplying a continuation state parameter to retrieve the remainder of the response. The transaction illustrates:

1. SDP client to SDP server: `SDP_ServiceSearchAttributeRequest`, specifying the generic `SynchronisationServiceClassID` (represented as a data element whose 32-bit UUID value is `sss . . . sss`) as the only element of the `ServiceSearchPattern`. The `SynchronisationServiceClassID` is assumed to be a 32-bit UUID. The requested attributes are the `ServiceRecordHandle` (attribute ID 0x0000), `ServiceClassIDList` (attribute ID 0x0001), `IconURL` (attribute ID 0x000C), `ServiceName` (attribute ID 0x0100), `ServiceDescription` (attribute ID 0x0101), and `ProviderName` (attribute ID 0x0102) attributes; as well as the service-specific `SupportedDataStores` (AttributeID 0x0301). Since the first two attribute IDs (0x0000 and 0x0001) and three other attribute IDs (0x0100, 0x0101, and 0x0102) are consecutive, they are specified as attribute ranges. The `TransactionID` is illustrated as `vvvv` to distinguish it from the `TransactionIDs` of the other Examples.

Note that values in the service record's primary language are requested for the text attributes (`ServiceName`, `ServiceDescription` and `ProviderName`) so that absolute attribute IDs may be used, rather than adding offsets to a base obtained from the `LanguageBaseAttributeIDList` attribute.

2. SDP server to SDP client: `SDP_ServiceSearchAttributeResponse`, returning the specified attributes for each of the three synchronization services. In the example, each `ServiceClassIDList` is assumed to contain a single element, the generic `SynchronisationServiceClassID` (a 32-bit UUID represented as `sss...sss`). Each of the other attributes contain illustrative data in the example (the strings have illustrative text; the icon URLs are illustrative, for each of the respective three synchronization services; and the `SupportedDataStore` attribute is represented as an unsigned 8-bit integer where 0x01 = vCard2.1, 0x02 = vCard3.0, 0x03 = vCal1.0 and 0x04 = iCal). Note that one of the service records (the third for which data is returned) has no `ServiceDescription` attribute. The attributes are returned as a data element sequence, where each element is in turn a data element sequence repre-

sending a list of attributes. Within each attribute list, the ServiceClassIDList is a data element sequence while the remaining attributes are single data elements. The Transaction ID is the same value as supplied by the SDP client in the corresponding request (0000). Since the entire result cannot be returned in a single response, a non-null continuation state is returned in this first response.

Note that the total length of the initial data element sequence (487 in the example) is indicated in the first response, even though only a portion of this data element sequence (368 bytes in the example, as indicated in the AttributeLists byte count) is returned in the first response. The remainder of this data element sequence is returned in the second response (without an additional data element header).

3. SDP client to SDP server: SDP_ServiceSearchAttributeRequest, with the same parameters as in step 1, except that the continuation state received from the server in step 2 is included as a request parameter. The TransactionID is changed to 0000 to distinguish it from previous request.
4. SDP server to SDP client: SDP_ServiceSearchAttributeResponse, with the remainder of the result computed in step 2 above. Since all of the remaining result fits in this second response, a null continuation state is included.

```

/* Part 1 -- Sent from SDP Client to SDP server */
SdpSDP_ServiceSearchAttributeRequest[33] {
    PDUID[1] {
        0x06
    }
    TransactionID[2] {
        0xvvvv
    }
    ParameterLength[2] {
        0x001B
    }
    ServiceSearchPattern[7] {
        DataElementSequence[7] {
            0b00110 0b101 0x05
            UUID[5] {
                /* SynchronisationServiceClassID */
                0b00011 0b010 0xssssssss
            }
        }
    }
    MaximumAttributeByteCount[2] {
        0x0190
    }
    AttributeIDList[18] {
        DataElementSequence[18] {
            0b00110 0b101 0x10
            AttributeIDRange[5] {
                0b00001 0b010 0x00000001
            }
            AttributeID[3] {
                0b00001 0b001 0x000C
            }
        }
    }
}

```

*Service Discovery Protocol***Bluetooth.**

```

    }
    AttributeIDRange[5] {
        0b00001 0b010 0x01000102
    }
    AttributeID[3] {
        0b00001 0b001 0x0301
    }
}
ContinuationState[1] {
    /* no continuation state */
    0x00
}
}

/* Part 2 -- Sent from SDP server to SDP client */
SdpSDP_ServiceSearchAttributeResponse[384] {
    PDUID[1] {
        0x07
    }
    TransactionID[2] {
        0xvvvv
    }
    ParameterLength[2] {
        0x017B
    }
    AttributeListByteCount[2] {
        0x0170
    }
    AttributeLists[368] {
        DataElementSequence[487] {
            0b00110 0b110 0x01E4
            DataElementSequence[178] {
                0b00110 0b101 0xB0
                Attribute[8] {
                    AttributeID[3] {
                        0b00001 0b001 0x0000
                    }
                    AttributeValue[5] {
                        /* service record handle */
                        0b00001 0b010 0xhhhhhhhh
                    }
                }
                Attribute[10] {
                    AttributeID[3] {
                        0b00001 0b001 0x0001
                    }
                }
            }
            AttributeValue[7] {
                DataElementSequence[7] {
                    0b00110 0b101 0x05
                    UUID[5] {
                        /* SynchronisationServiceClassID */
                        0b00011 0b010 0xssssssss
                    }
                }
            }
        }
        Attribute[35] {

```

*Service Discovery Protocol***Bluetooth.**

```

        AttributeID[3] {
            0b00001 0b001 0x000C
        }
        AttributeValue[32] {
            /* IconURL; '*' replaced by client application */
            0b01000 0b101 0x1E
            "http://Synchronisation/icons/*"
        }
    }
    Attribute[22] {
        AttributeID[3] {
            0b00001 0b001 0x0100
        }
        AttributeValue[19] {
            /* service name */
            0b00100 0b101 0x11
            "Address Book Sync"
        }
    }
    Attribute[59] {
        AttributeID[3] {
            0b00001 0b001 0x0101
        }
        AttributeValue[56] {
            /* service description */
            0b00100 0b101 0x36
            "Synchronisation Service for"
            " vCard Address Book Entries"
        }
    }
    Attribute[37] {
        AttributeID[3] {
            0b00001 0b001 0x0102
        }
        AttributeValue[34] {
            /* service provider */
            0b00100 0b101 0x20
            "Synchronisation Specialists Inc."
        }
    }
    Attribute[5] {
        AttributeID[3] {
            0b00001 0b001 0x0301
        }
        AttributeValue[2] {
            /* Supported Data Store 'phonebook' */
            0b00001 0b000 0x01
        }
    }
}
DataElementSequence[175] {
    0b00110 0b101 0xAD
    Attribute[8] {
        AttributeID[3] {
            0b00001 0b001 0x0000
        }
        AttributeValue[5] {

```

*Service Discovery Protocol***Bluetooth.**

```

        /* service record handle */
        0b00001 0b010 0xmmmmmmmm
    }
}
Attribute[10] {
    AttributeID[3] {
        0b00001 0b001 0x0001
    }
    AttributeValue[7] {
        DataElementSequence[7] {
            0b00110 0b101 0x05
            UUID[5] {
                /* SynchronisationServiceClassID */
                0b00011 0b010 0xssssssss
            }
        }
    }
}
Attribute[35] {
    AttributeID[3] {
        0b00001 0b001 0x000C
    }
    AttributeValue[32] {
        /* IconURL; '*' replaced by client application */
        0b01000 0b101 0x1E
        "http://Synchronisation/icons/*"
    }
}
Attribute[21] {
    AttributeID[3] {
        0b00001 0b001 0x0100
    }
    AttributeValue[18] {
        /* service name */
        0b00100 0b101 0x10
        "Appointment Sync"
    }
}
Attribute[57] {
    AttributeID[3] {
        0b00001 0b001 0x0101
    }
    AttributeValue[54] {
        /* service description */
        0b00100 0b101 0x34
        "Synchronisation Service for"
        " vCal Appointment Entries"
    }
}
Attribute[37] {
    AttributeID[3] {
        0b00001 0b001 0x0102
    }
    AttributeValue[34] {
        /* service provider */
        0b00100 0b101 0x20
        "Synchronisation Specialists Inc."
    }
}

```

*Service Discovery Protocol***Bluetooth.**

```

    }
  }
  Attribute[5] {
    AttributeID[3] {
      0b00001 0b001 0x0301
    }
    AttributeValue[2] {
      /* Supported Data Store 'calendar' */
      0b00001 0b000 0x03
    }
  }
}
/* } Data element sequence of attribute lists */
/* is not completed in this PDU. */
}
ContinuationState[9] {
  /* 8 bytes of continuation state */
  0x08 0xzzzzzzzzzzzzzzzz
}
}

/* Part 3 -- Sent from SDP Client to SDP server */
SdpSDP_ServiceSearchAttributeRequest[41] {
  PDUID[1] {
    0x06
  }
  TransactionID[2] {
    0xwww
  }
  ParameterLength[2] {
    0x0024
  }
  ServiceSearchPattern[7] {
    DataElementSequence[7] {
      0b00110 0b101 0x05
      UUID[5] {
        /* SynchronisationServiceClassID */
        0b00011 0b010 0xssssssss
      }
    }
  }
  MaximumAttributeByteCount[2] {
    0x0180
  }
  AttributeIDList[18] {
    DataElementSequence[18] {
      0b00110 0b101 0x10
      AttributeIDRange[5] {
        0b00001 0b010 0x00000001
      }
      AttributeID[3] {
        0b00001 0b001 0x000C
      }
      AttributeIDRange[5] {
        0b00001 0b010 0x01000102
      }
      AttributeID[3] {

```

*Service Discovery Protocol***Bluetooth.**

```

        0b00001 0b001 0x0301
    }
}
ContinuationState[9] {
    /* same 8 bytes of continuation state */
    /* received in part 2 */
    0x08 0xzzzzzzzzzzzzzzzz
}
}

```

Part 4 -- Sent from SDP server to SDP client

```

SdpSDP_ServiceSearchAttributeResponse[115] {
    PDUID[1] {
        0x07
    }
    TransactionID[2] {
        0xwww
    }
    ParameterLength[2] {
        0x006E
    }
    AttributeListByteCount[2] {
        0x006B
    }
    AttributeLists[107] {
        /* Continuing the data element sequence of */
        /* attribute lists begun in Part 2. */
        DataElementSequence[107] {
            0b00110 0b101 0x69
            Attribute[8] {
                AttributeID[3] {
                    0b00001 0b001 0x0000
                }
                AttributeValue[5] {
                    /* service record handle */
                    0b00001 0b010 0xffffffff
                }
            }
            Attribute[10] {
                AttributeID[3] {
                    0b00001 0b001 0x0001
                }
                AttributeValue[7] {
                    DataElementSequence[7] {
                        0b00110 0b101 0x05
                        UUID[5] {
                            /* SynchronisationServiceClassID */
                            0b00011 0b010 0xssssssss
                        }
                    }
                }
            }
            Attribute[35] {
                AttributeID[3] {
                    0b00001 0b001 0x000C
                }
            }
        }
    }
}

```

*Service Discovery Protocol***Bluetooth.**

```

    }
    AttributeValue[32] {
        /* IconURL; '*' replaced by client application */
        0b01000 0b101 0x1E
        "http://DevManufacturer/icons/"
    }
}
Attribute[18] {
    AttributeID[3] {
        0b00001 0b001 0x0100
    }
    AttributeValue[15] {
        /* service name */
        0b00100 0b101 0x0D
        "Calendar Sync"
    }
}
Attribute[29] {
    AttributeID[3] {
        0b00001 0b001 0x0102
    }
    AttributeValue[26] {
        /* service provider */
        0b00100 0b101 0x18
        "Device Manufacturer Inc."
    }
}
Attribute[5] {
    AttributeID[3] {
        0b00001 0b001 0x0301
    }
    AttributeValue[2] {
        /* Supported Data Store 'calendar' */
        0b00001 0b000 0x03
    }
}
/* This completes the data element sequence */
/* of attribute lists begun in Part 2.
}
ContinuationState[1] {
    /* no continuation state */
    0x00
}
}

```


Part F:1

RFCOMM with TS 07.10

Serial Port Emulation

This document specifies the RFCOMM protocol by specifying a subset of the ETSI TS 07.10 standard, along with some Bluetooth-specific adaptations

CONTENTS

1	Introduction	389
1.1	Overview	389
1.2	Device Types	389
1.3	Byte Ordering	390
2	RFCOMM Service Overview	391
2.1	RS-232 Control Signals.....	391
2.2	Null Modem Emulation	391
2.3	Multiple Emulated Serial Ports.....	393
2.3.1	Multiple Emulated Serial Ports between two Devices	393
2.3.2	Multiple Emulated Serial Ports and Multiple BT Devices.....	393
3	Service Interface Description.....	395
3.1	Service Definition Model	395
4	TS 07.10 Subset Supported by RFCOMM	396
4.1	Options and Modes	396
4.2	Frame Types	396
4.3	Commands.....	396
4.4	Convergence Layers.....	397
5	TS 07.10 Adaptations for RFCOMM	398
5.1	Media Adaptation	398
5.1.1	FCS calculation	398
5.2	TS 07.10 Multiplexer Start-up and Closedown Procedure	399
5.2.1	Start-up procedure	399
5.2.2	Close-down procedure	399
5.2.3	Link loss handling.....	399
5.3	System Parameters.....	400
5.4	DLCI allocation with RFCOMM server channels.....	400
5.5	Multiplexer Control Commands.....	401
5.5.1	Remote Port Negotiation Command (RPN)	401
5.5.2	Remote Line Status Command (RLS).....	402
5.5.3	DLC parameter negotiation (PN).....	402
6	Flow Control	403
6.1	L2CAP Flow Control in Overview.....	403
6.2	Wired Serial Port Flow Control.....	403
6.3	RFCOMM Flow Control.....	403
6.4	Port Emulation Entity Serial Flow Control	403

*RFCOMM with TS 07.10***Bluetooth.**

7	Interaction with Other Entities	405
7.1	Port Emulation and Port Proxy Entities.....	405
7.1.1	Port Emulation Entity.....	405
7.1.2	Port Proxy Entity	405
7.2	Service Registration and Discovery	405
7.3	Lower Layer Dependencies	407
7.3.1	Reliability.....	407
7.3.2	Low power modes	407
8	References.....	408
9	Terms and Abbreviations	409

1 INTRODUCTION

The RFCOMM protocol provides emulation of serial ports over the L2CAP protocol. The protocol is based on the ETSI standard TS 07.10. This document does not contain a complete specification. Instead, references are made to the relevant parts of the TS 07.10 standard. Only a subset of the TS 07.10 standard is used, and some adaptations of the protocol are specified in this document.

1.1 OVERVIEW

RFCOMM is a simple transport protocol, with additional provisions for emulating the 9 circuits of RS-232 (EIA/TIA-232-E) serial ports:

The RFCOMM protocol supports up to 60 simultaneous connections between two BT devices. The number of connections that can be used simultaneously in a BT device is implementation-specific.

1.2 DEVICE TYPES

For the purposes of RFCOMM, a complete communication path involves two applications running on different devices (the communication endpoints) with a communication segment between them. Figure 1.1 shows the complete communication path. (In this context, the term *application* may mean other things than end-user application; e.g. higher layer protocols or other services acting on behalf of end-user applications.)

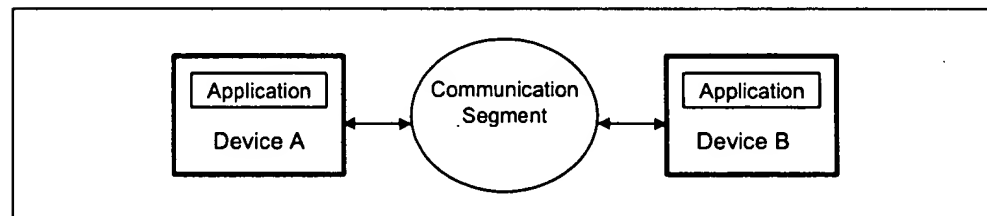


Figure 1.1: RFCOMM Communication Segment

RFCOMM is intended to cover applications that make use of the serial ports of the devices in which they reside. In the simple configuration, the communication segment is a BT link from one device to another (direct connect), see Figure 1.2. Where the communication segment is another network, BT is used for the path between the device and a network connection device like a modem. RFCOMM is only concerned with the connection between the devices in the direct connect case, or between the device and a modem in the network case. RFCOMM can support other configurations, such as modules that communicate via BT on one side and provide a wired interface on the other side, as shown in Figure 1.3. These devices are not really modems but offer a similar service. They are therefore not explicitly discussed here.

Bluetooth.

Basically two device types exist that RFCOMM must accommodate. Type 1 devices are communication end points such as computers and printers. Type 2 devices are those that are part of the communication segment; e.g. modems. Though RFCOMM does not make a distinction between these two device types in the protocol, accommodating both types of devices impacts the RFCOMM protocol.

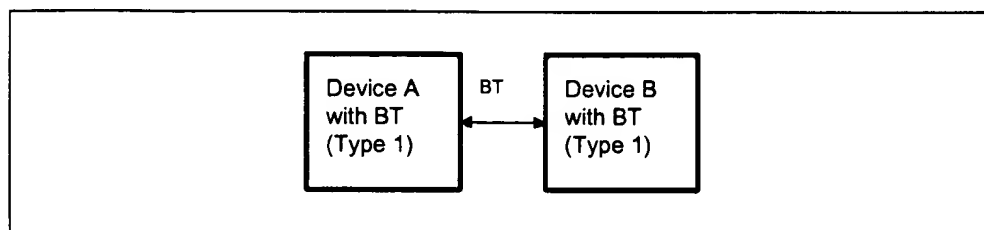


Figure 1.2: RFCOMM Direct Connect

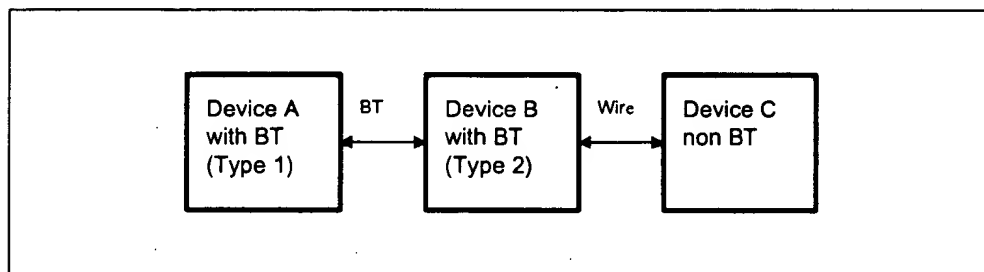


Figure 1.3: RFCOMM used with legacy COMM device

The information transferred between two RFCOMM entities has been defined to support both type 1 and type 2 devices. Some information is only needed by type 2 devices while other information is intended to be used by both. In the protocol, no distinction is made between type 1 and type 2. It is therefore up to the RFCOMM implementers to determine if the information passed in the RFCOMM protocol is of use to the implementation. Since the device is not aware of the type of the other device in the communication path, each must pass on all available information specified by the protocol.

1.3 BYTE ORDERING

This document uses the same byte ordering as the TS 07.10 specification; i.e. all binary numbers are in Least Significant Bit to Most Significant Bit order, reading from left to right.

2 RFCOMM SERVICE OVERVIEW

RFCOMM emulates RS-232 (EIA/TIA-232-E) serial ports. The emulation includes transfer of the state of the non-data circuits. RFCOMM has a built-in scheme for null modem emulation.

In the event that a baud rate is set for a particular port through the RFCOMM service interface, that will not affect the actual data throughput in RFCOMM; i.e. RFCOMM does not incur artificial rate limitation or pacing. However, if either device is a type 2 device (relays data onto other media), or if data pacing is done above the RFCOMM service interface in either or both ends, actual throughput will, on an average, reflect the baud rate setting.

RFCOMM supports emulation of multiple serial ports between two devices and also emulation of serial ports between multiple devices, see Section 2.3 on page 393.

2.1 RS-232 CONTROL SIGNALS

RFCOMM emulates the 9 circuits of an RS-232 interface. The circuits are listed below.

Pin	Circuit Name
102	Signal Common
103	Transmit Data (TD)
104	Received Data (RD)
105	Request to Send (RTS)
106	Clear to Send (CTS)
107	Data Set Ready (DSR)
108	Data Terminal Ready (DTR)
109	Data Carrier Detect (CD)
125	Ring Indicator (RI)

Table 2.1: Emulated RS-232 circuits in RFCOMM

2.2 NULL MODEM EMULATION

RFCOMM is based on TS 07.10. When it comes to transfer of the states of the non-data circuits, TS 07.10 does not distinguish between DTE and DCE devices. The RS-232 control signals are sent as a number of DTE/DCE independent signals, see Table 2.2.

TS 07.10 Signals	Corresponding RS-232 Control Signals
RTC	DSR, DTR
RTR	RTS, CTS
IC	RI
DV	DCD

Table 2.2: TS 07.10 Serial Port Control Signals

The way in which TS 07.10 transfers the RS-232 control signals creates an implicit null modem when two devices of the same kind are connected together. Figure 2.1 shows the null modem that is created when two DTE are connected via RFCOMM. No single null-modem cable wiring scheme works in all cases; however the null modem scheme provided in RFCOMM should work in most cases.

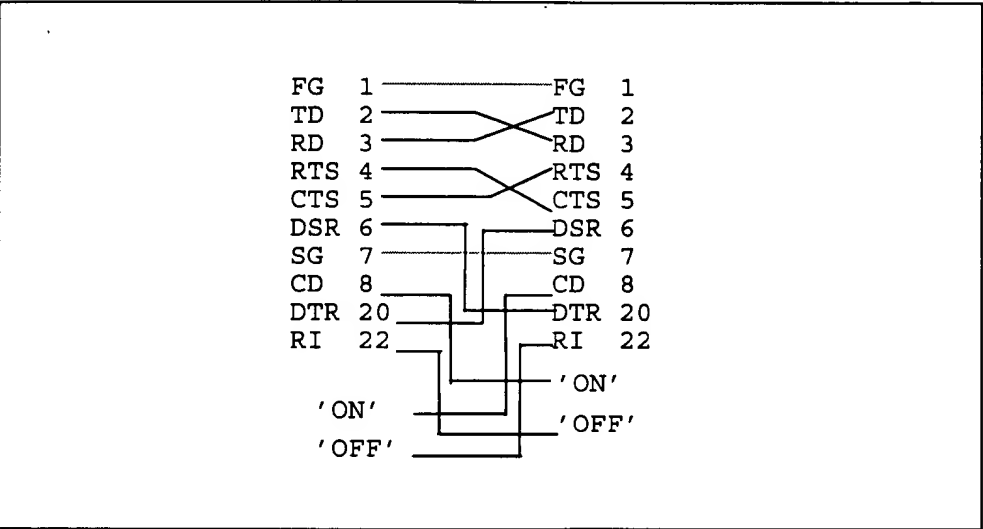


Figure 2.1: RFCOMM DTE-DTE Null Modem Emulation

2.3 MULTIPLE EMULATED SERIAL PORTS

2.3.1 Multiple Emulated Serial Ports between two Devices

Two BT devices using RFCOMM in their communication may open multiple emulated serial ports. RFCOMM supports up to 60 open emulated ports; however the number of ports that can be used in a device is implementation-specific.

A Data Link Connection Identifier (DLCI) [1] identifies an ongoing connection between a client and a server application. The DLCI is represented by 6 bits, but its usable value range is 2...61; in TS 07.10, DLCI 0 is the dedicated control channel, DLCI 1 is unusable due to the concept of Server Channels, and DLCI 62-63 is reserved. The DLCI is unique within one RFCOMM session between two devices. (This is explained further in Section 2.3.2) To account for the fact that both client and server applications may reside on both sides of an RFCOMM session, with clients on either side making connections independent of each other, the DLCI value space is divided between the two communicating devices using the concept of RFCOMM server channels. This is further described in Section 5.4.

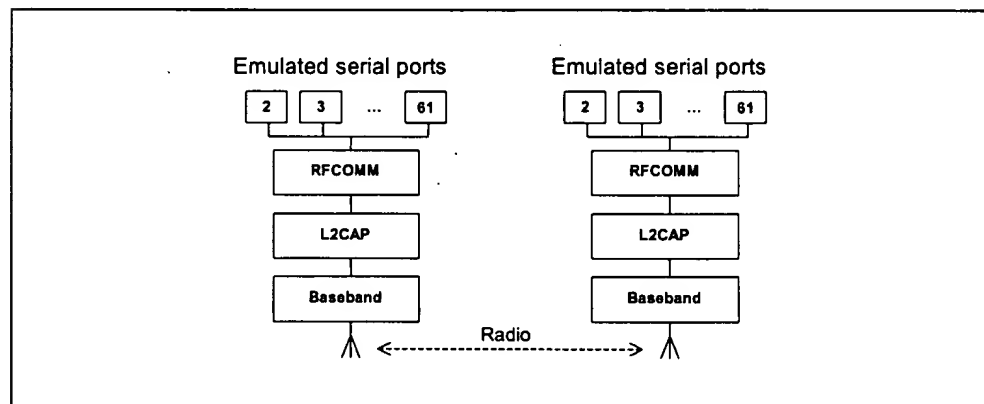


Figure 2.2: Multiple Emulated Serial Ports.

2.3.2 Multiple Emulated Serial Ports and Multiple BT Devices

If a BT device supports multiple emulated serial ports and the connections are allowed to have endpoints in different BT devices, then the RFCOMM entity must be able to run multiple TS 07.10 multiplexer sessions, see Figure 2.3. Note that each multiplexer session is using its own L2CAP channel ID (CID). The ability to run multiple sessions of the TS 07.10 multiplexer is optional for RFCOMM.

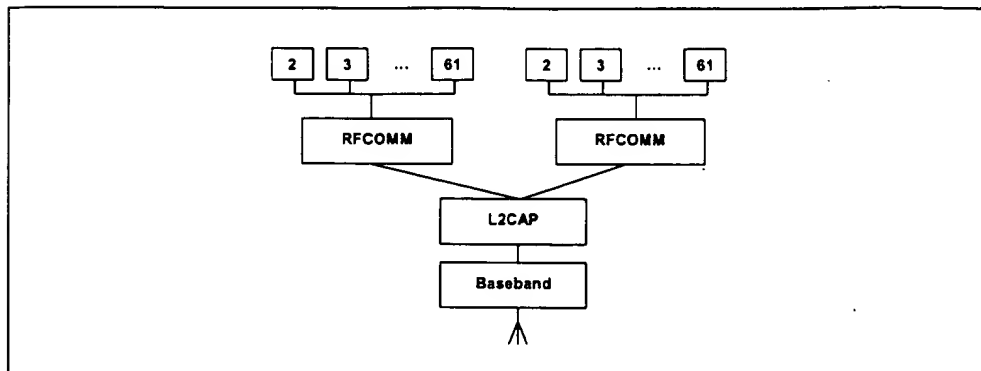


Figure 2.3: Emulating serial ports coming from two BT devices.

3 SERVICE INTERFACE DESCRIPTION

RFCOMM is intended to define a protocol that can be used to emulate serial ports. In most systems, RFCOMM will be part of a port driver which includes a serial port emulation entity.

3.1 SERVICE DEFINITION MODEL

The figure below shows a model of how RFCOMM fits into a typical system. This figure represents the RFCOMM reference model.

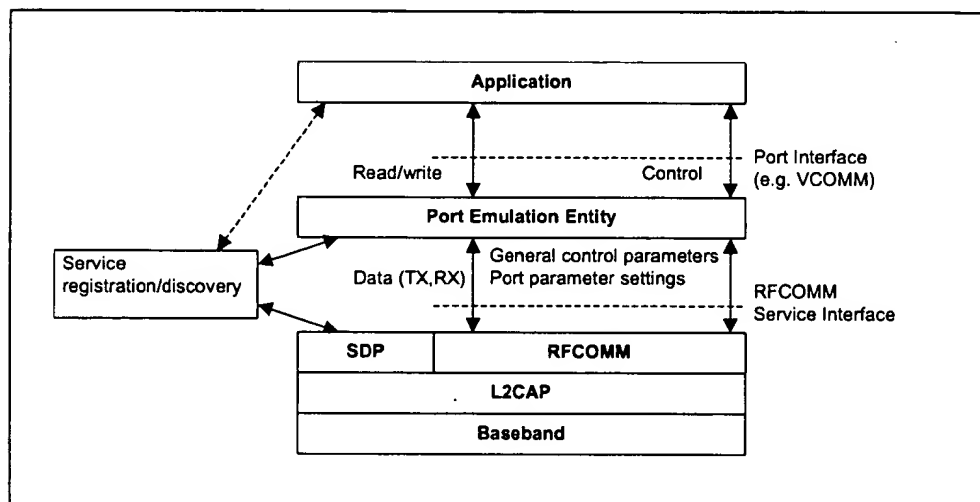


Figure 3.1: RFCOMM reference model

The elements of the RFCOMM reference model are described below.

Element	Description
Application	Applications that utilize a serial port communication interface
Port Emulation Entity	The port emulation entity maps a system-specific communication interface (API) to the RFCOMM services. The port emulation entity plus RFCOMM make up a port driver
RFCOMM	Provides a transparent data stream and control channel over an L2CAP channel. Multiplexes multiple emulated serial ports
Service Registration/ Discovery	Server applications register here on local device, and it provides services for client applications to discover how to reach server applications on other devices.
L2CAP	Protocol multiplexing, SAR
Baseband	Baseband protocols defined by BT

4 TS 07.10 SUBSET SUPPORTED BY RFCOMM

4.1 OPTIONS AND MODES

RFCOMM uses the basic option of TS 07.10.

4.2 FRAME TYPES

Table 4.1 shows the TS 7.10 frame types that are supported in RFCOMM.

Frame Types
Set Asynchronous Balanced Mode (SABM) command
Unnumbered Acknowledgement (UA) response
Disconnected Mode (DM) response
Disconnect (DISC) command
Unnumbered information with header check (UIH) command and response

Table 4.1: Supported frame types in RFCOMM

The 'Unnumbered Information (UI) command and response' are not supported by RFCOMM. Since the error recovery mode option of the TS 07.10 protocol is not used in RFCOMM none of the associated frame types are supported.

4.3 COMMANDS

TS 07.10 defines a multiplexer that has a dedicated control channel, DLCI 0. The control channel is used to convey information between two multiplexers. The following commands in TS 07.10 are supported by RFCOMM:

Supported Control Channel Commands
Test Command (Test)
Flow Control On Command (Fcon)
Flow Control Off Command (Fcoff)
Modem Status Command (MSC)
Remote Port Negotiation Command (RPN)
Remote Line Status (RLS)
DLC parameter negotiation (PN)
Non Supported Command Response (NSC)

Whenever a non-supported command type is received a 'Non-Supported Command Response (NSC)' should be sent.

4.4 CONVERGENCE LAYERS

RFCOMM only supports the type 1 convergence layer in TS 07.10.

The Modem Status Command (MSC) shall be used to convey the RS-232 control signals and the break signal for all emulated serial ports.

5 TS 07.10 ADAPTATIONS FOR RFCOMM

5.1 MEDIA ADAPTATION

The opening flag and the closing flags in the 07.10 basic option frame are not used in RFCOMM, instead it is only the fields contained between the flags that are exchanged between the L2CAP layer and RFCOMM layer, see Figure 5.1.

Flag	Address	Control	Length Indicator	Information	FCS	Flag
0111 1101	1 octet	1 octet	1 or 2 octets	Unspecified length but integral number of octets	1 octet	0111 1101

Figure 5.1: Frame Structure for Basic option. Note that the opening and closing flags from the 07.10 Basic option are excluded in RFCOMM.

5.1.1 FCS calculation

In 07.10, the frame check sequence (FCS) is calculated on different sets of fields for different frame types. These are the fields that the FCS are calculated on:

For SABM, DISC, UA, DM frames: on Address, Control and length field.

For UIH frames: on Address and Control field.

(This is stated here for clarification, and to set the standard for RFCOMM; the fields included in FCS calculation have actually changed in version 7.0.0 of TS 07.10, but RFCOMM will not change the FCS calculation scheme from the one above.)

5.2 TS 07.10 MULTIPLEXER START-UP AND CLOSEDOWN PROCEDURE

The start-up and closedown procedures as specified in section 5.7 in TS 07.10 are not supported. This means that the AT-command AT+CMUX is not supported by RFCOMM, neither is the multiplexer close down (CLD) command.

At any time, there must be at most one RFCOMM session between any pair of devices. When establishing a new DLC, the initiating entity must check if there already exists an RFCOMM session with the remote device, and if so, establish the new DLC on that. A session is identified by the Bluetooth BD_ADDR of the two endpoints¹.

5.2.1 Start-up procedure

The device opening up the first emulated serial port connection between two devices is responsible for first establishing the multiplexer control channel. This involves the following steps:

- Establish an L2CAP channel to the peer RFCOMM entity, using L2CAP service primitives, see L2CAP "Service Primitives" on page 295.
- Start the RFCOMM multiplexer by sending SABM command on DLCI 0, and await UA response from peer entity. (Further optional negotiation steps are possible.)

After these steps, DLCs for user data traffic can be established.

5.2.2 Close-down procedure

The device closing the last connection (DLC) on a particular session is responsible for closing the multiplexer by closing the corresponding L2CAP channel.

Closing the multiplexer by first sending a DISC command frame on DLCI 0 is optional, but it is mandatory to respond correctly to a DISC (with UA response).

5.2.3 Link loss handling

If an L2CAP link loss notification is received, the local RFCOMM entity is responsible for sending a connection loss notification to the port emulation/proxy entity for each active DLC. Then all resources associated with the RFCOMM session should be freed.

The appropriate action to take in the port emulation/proxy entity depends on the API on top. For example, for an emulated serial port (vCOMM), it would be suitable to drop CD, DSR and CTS signals (assuming device is a DTE).

1. This implies that, when responding to an L2CAP connection indication, the RFCOMM entity should save and associate the new RFCOMM session with the remote BD_ADDR. This is, at least, necessary if subsequent establishment of a DLC in the opposite direction is possible (which may depend on device capabilities).

5.3 SYSTEM PARAMETERS

Table 5.1 contains all the applicable system parameters for the RFCOMM implementation of the TS 07.10 multiplexer.

System Parameter	Value
Maximum Frame Size (<i>N1</i>)	Default: 127 (negotiable range 23 – 32767)
Acknowledgement Timer (<i>T1</i>)	60 seconds
Response Timer for Multiplexer Control Channel (<i>T2</i>)	60 seconds

Table 5.1: System parameter values

Note: The timer *T1* is the timeout for *frames* sent with the P/F-bit set to one (this applies only to SABM and DISC frames in RFCOMM). *T2* is the timeout for *commands* sent in UIH frames on DLCI 0.

Since RFCOMM relies on lower layers to provide reliable transmission, the default action performed on timeouts is to close down the multiplexer session. The only exception to this is when trying to set up a new DLC on an existing session; i.e. waiting for the UA response for a SABM command. In this case, the initiating side may defer the timeout by an unspecified amount of time if it has knowledge that the delay is due to user interaction (e.g. authentication procedure in progress). When/if the connection attempt is eventually considered to have timed out, the initiating side must send a DISC command frame on the same DLCI as the original SABM command frame, in order to notify the other party that the connection attempt is aborted. (After that the initiating side will, as usual, expect a UA response for the DISC command.)

5.4 DLCI ALLOCATION WITH RFCOMM SERVER CHANNELS

To account for the fact that both client and server applications may reside on both sides of an RFCOMM session, with clients on either side making connections independent of each other, the DLCI value space is divided between the two communicating devices using the concept of RFCOMM server channels and a direction bit.

The RFCOMM server channel number is a subset of the bits in the DLCI part of the address field in the TS 07.10 frame.

Bit No.	1	2	3	4	5	6	7	8
TS 07.10	EA	C/R	DLCI					
RFCOMM	EA	C/R	D	Server Channel				

Table 5.2: The format of the Address Field

Server applications registering with an RFCOMM service interface are assigned a Server Channel number in the range 1...30. [0 and 31 should not be used since the corresponding DLCIs are reserved in TS 07.10] It is this value that should be registered in the Service Discovery Database, see Section 7.2.

For an RFCOMM session, the initiating device is given the direction bit D=1 (and conversely, D=0 in the other device). When establishing a new data link connection on an existing RFCOMM session, the direction bit is used in conjunction with the Server Channel to determine the DLCI to use to connect to a specific application. This DLCI is thereafter used for all packets in both directions between the endpoints.

In effect, this partitions the DLCI value space such that server applications on the non-initiating device are reachable on DLCIs 2,4,6,...,60; and server applications on the initiating device are reachable on DLCIs 3,5,7,...,61. (Note that for a device that supports multiple simultaneous RFCOMM sessions to two or more devices, the direction bit might not be the same on all sessions.)

An RFCOMM entity making a new DLC on an existing session forms the DLCI by combining the Server Channel for the application on the other device, and the inverse of its own direction bit for the session.

DLCI 1 and 62-63 are reserved and never used in RFCOMM.

5.5 MULTIPLEXER CONTROL COMMANDS

Note that in TS 07.10, some Multiplexer Control commands pertaining to specific DLCIs may be exchanged on the control channel (DLCI 0) *before* the corresponding DLC has been established. (This refers the PN and RPN commands.) All such states associated with an individual DLC must be reset to their default values upon receiving a DISC command frame, or when closing the DLC from the local side. This is to ensure that all DLC (re-)establishments on the same session will have predictable results, irrespective of the session history.

5.5.1 Remote Port Negotiation Command (RPN)

The RPN command can be used before a new DLC is opened and should be used whenever the port settings change.

The RPN command is specified as optional in TS 07.10, but it is mandatory to recognize and respond to it in RFCOMM. (Although the handling of individual settings are implementation-dependent.)

5.5.2 R mote Line Status Command (RLS)

This command is used for indication of remote port line status.

The RLS command is specified as optional in TS 07.10, but it is mandatory to recognize and respond to it in RFCOMM. (Although the handling of individual settings are implementation-dependent.)

5.5.3 DLC parameter negotiation (PN)

The PN command is specified as optional in TS 07.10, but it is mandatory to recognize and respond to it in RFCOMM. This command can be used before a new DLC is opened.

There are some parameters in the PN command which convey information not applicable to RFCOMM. These fields must therefore be set to predetermined values by the sender, and they must be ignored by the receiver. This concern the following fields (see table 3 in ref. [1]):

- I1-I4 must be set to 0. (Meaning: use UIH frames.)
- CL1-CL4 must be set to 0. (Meaning: use convergence layer type 1.)
- T1-T8 must be set to 0. (Meaning: acknowledgment timer *T1*, which is not negotiable in RFCOMM.)
- NA1-NA8 must be set to 0. (Meaning: number of retransmissions *N2*; always 0 for RFCOMM)
- K1-K3 must be set to 0. (Meaning: defines the window size for error recovery mode, which is not used for RFCOMM.)

If a command is received with invalid (or for some reason unacceptable) values in any field, a DLC parameter negotiation response must be issued with values that are acceptable to the responding device.

6 FLOW CONTROL

Wired ports commonly use flow control such as RTS/CTS to control communications. On the other hand, the flow control between RFCOMM and the lower layer L2CAP depends on the service interface supported by the implementation. In addition RFCOMM has its own flow control mechanisms. This section describes the different flow control mechanisms.

6.1 L2CAP FLOW CONTROL IN OVERVIEW

L2CAP relies on the flow control mechanism provided by the Link Manager layer in the baseband. The flow control mechanism between the L2CAP and RFCOMM layers is implementation-specific.

6.2 WIRED SERIAL PORT FLOW CONTROL

Wired Serial ports falls into two camps – software flow control using characters such as XON/XOFF, and flow control using RTS/CTS or DTR/DSR circuits. These methods may be used by both sides of a wired link, or may be used only in one direction.

6.3 RFCOMM FLOW CONTROL

The RFCOMM protocol provides two flow control mechanisms:

1. The RFCOMM protocol contains flow control commands that operate on the aggregate data flow between two RFCOMM entities; i.e. all DLCIs are affected. The control channel commands, FCon and FCoff, are defined in section 5.4.6.3 in ref [1].
2. The Modem Status command as defined in section 5.4.6.3 in ref [1] is the flow control mechanism that operates on individual DLCI.

6.4 PORT EMULATION ENTITY SERIAL FLOW CONTROL

On Type 1 devices some port drivers (Port Emulation Entities plus RFCOMM) will need to provide flow control services as specified by the API they are emulating. An application may request a particular flow control mechanism like XON/XOFF or RTS/CTS and expect the port driver to handle the flow control. On type 2 devices the port driver may need to perform flow control on the non-RFCOMM part of the communication path; i.e. the physical RS-232 port. This flow control is specified via the control parameters sent by the peer RFCOMM entity (usually a type 1 device). The description of flow control in this section is for port drivers on type 1 devices.

Since RFCOMM already has its own flow control mechanism, the port driver does not need to perform flow control using the methods requested by the application. In the ideal case, the application sets a flow control mechanism

and assumes that the COMM system will handle the details. The port driver could then simply ignore the request and rely on RFCOMM's flow control. The application is able to send and receive data, and does not know or care that the port driver did not perform flow control using the mechanism requested. However, in the real world some problems arise.

- The RFCOMM-based port driver is running on top of a packet-based protocol where data may be buffered somewhere in the communication path. Thus, the port driver cannot perform flow control with the same precision as in the wired case.
- The application may decide to apply the flow control mechanism itself in addition to requesting flow control from the port driver.

These problems suggest that the port driver must do some additional work to perform flow control emulation properly. Here are the basic rules for flow control emulation.

- The port driver will not solely rely on the mechanism requested by the application but use a combination of flow control mechanisms.
- The port driver must be aware of the flow control mechanisms requested by the application and behave like the wired case when it sees changes on the non-data circuits (hardware flow control) or flow control characters in the incoming data (software flow control). For example, if XOFF and XON characters would have been stripped in the wired case they must be stripped by the RFCOMM based port driver.
- If the application sets a flow control mechanism via the port driver interface and then proceeds to invoke the mechanism on its own, the port driver must behave in a manner similar to that of the wired case (e.g. If XOFF and XON characters would have been passed through to the wire in the wired case the port driver must also pass these characters).

These basic rules are applied to emulate each of the wired flow control schemes. Note that multiple types of flow control can be set at the same time. Section 5.4.8 in ref [1] defines each flow control mechanism.

7 INTERACTION WITH OTHER ENTITIES

7.1 PORT EMULATION AND PORT PROXY ENTITIES

This section defines how the RFCOMM protocol should be used to emulate serial ports. Figure 7.1 shows the two device types that the RFCOMM protocol supports.

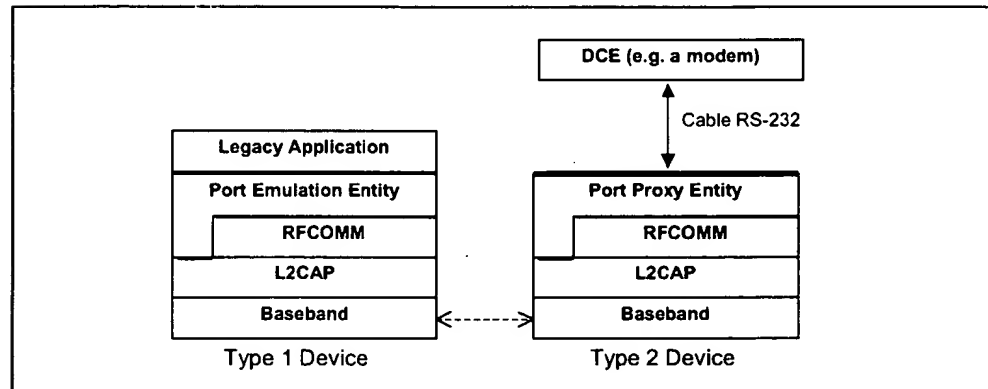


Figure 7.1: The RFCOMM communication model

Type 1 devices are communication endpoints such as computers and printers. Type 2 devices are part of a communication segment; e.g. modems.

7.1.1 Port Emulation Entity

The port emulation entity maps a system specific communication interface (API) to the RFCOMM services.

7.1.2 Port Proxy Entity

The port proxy entity relays data from RFCOMM to an external RS-232 interface linked to a DCE. The communications parameters of the RS-232 interface are set according to received RPN commands, see Section 5.5.1.

7.2 SERVICE REGISTRATION AND DISCOVERY

Registration of individual applications or services, along with the information needed to reach those (i.e. the RFCOMM Server Channel) is the responsibility of each application respectively (or possibly a Bluetooth configuration application acting on behalf of legacy applications not directly aware of Bluetooth).

Below is a template/example for developing service records for a given service or profile using RFCOMM. It illustrates the inclusion of the ServiceClassList with a single service class, a ProtocolDescriptor List with two protocols

(although there may be more protocols on top of RFCOMM). The example shows the use of one other universal attribute (ServiceName). For each service running on top of RFCOMM, appropriate SDP-defined universal attributes and/or service-specific attributes will apply. For additional information on Service Records, see the SDP Specification, Section 2.2 on page 332.

The attributes that a client application needs (at a minimum) to connect to a service on top of RFCOMM are the ServiceClassIDList and the ProtocolDescriptorList (corresponding to the shaded rows in the table below).

Item	Definition	Type/Size	Value	Attribute ID
ServiceClassIDList			Note1	0x0001
ServiceClass0	Note5	UUID/32-bit	Note1	
ProtocolDescriptorList				0x0004
Protocol0	L2CAP	UUID/32-bit	L2CAP /Note1	
Protocol1	RFCOMM	UUID/32-bit	RFCOMM /Note1	
ProtocolSpecificParameter0	Server Channel	Uint8	N = server channel #	
[other protocols]		UUID/32-bit	Note1	
[other protocol-specific parameters]	Note3	Note3	Note3	
ServiceName	Displayable text name	DataElement/ String	'Example service'	Note2
[other universal attributes as appropriate for this service]	Note4	Note4	Note4	Note4
[service-specific attributes]	Note3	Note3	Note3	Note3

Notes:

1. Defined in "Bluetooth Assigned Numbers" on page 1009.
2. For national language support for all 'displayable' text string attributes, an offset has to be added to the LanguageBaseAttributeIDList value for the selected language (see the SDP Specification, Section 5.1.14 on page 365 for details).
3. To be defined (where necessary) for the specific service.
4. For a specific service some of the SDP-defined universal attributes may apply. See the SDP Specification, Section 5.1 on page 358.
5. This indicates the class of service. It can be a single entry or a list of service classes ranging from generic to most specific.

7.3 LOWER LAYER DEPENDENCIES

7.3.1 Reliability

RFCOMM uses the services of L2CAP to establish L2CAP channels to RFCOMM entities on other devices. An L2CAP channel is used for the RFCOMM/TS 07.10 multiplexer session. On such a channel, the TS 07.10 frames listed in Section 4.2 are sent, with the adaptation defined in Section 5.1.

Some frame types (SABM and DISC) as well as UIH frames with multiplexer control commands sent on DLCI 0 always require a response from the remote entity, so they are acknowledged on the RFCOMM level (but not retransmitted in the absence of acknowledgment, see Section 5.3). Data frames do not require any response in the RFCOMM protocol, and are thus unacknowledged.

Therefore, RFCOMM must require L2CAP to provide channels with maximum reliability, to ensure that all frames are delivered in order, and without duplicates. Should an L2CAP channel fail to provide this, RFCOMM expects a link loss notification, which should be handled by RFCOMM as described in Section 5.2.3.

7.3.2 Low power modes

If all L2CAP channels towards a certain device are idle for a certain amount of time, a decision may be made to put that device in a low power mode (i.e. use hold, sniff or park, see 'Baseband Specification' Section 10.10.3 on page 125). This will be done without any interference from RFCOMM. RFCOMM can state its latency requirements to L2CAP. This information may be used by lower layers to decide which low power mode(s) to use.

The RFCOMM protocol as such does not suffer from latency delays incurred by low power modes, and consequentially, this specification does not state any maximum latency requirement on RFCOMM's behalf. Latency sensitivity inherently depends on application requirements, which suggests that an RFCOMM service interface implementation could include a way for applications to state latency requirements, to be aggregated and conveyed to L2CAP by the RFCOMM implementation. (That is if such procedures make sense for a particular platform.)

8 REFERENCES

- [1] TS 07.10, ver 6.3.0, ETSI
- [2] Bluetooth L2CAP Specification
- [3] Bluetooth SDP Specification
- [4] Bluetooth Assigned Numbers

9 TERMS AND ABBREVIATIONS

The following terms are used throughout the document.

DTE	Data Terminal Equipment – in serial communications, DTE refers to a device at the endpoint of the communications path; typically a computer or terminal
DCE	Data Circuit-Terminating Equipment – in serial communications, DCE refers to a device between the communication endpoints whose sole task is to facilitate the communications process; typically a modem
RFCOMM Initiator	The device initiating the RFCOMM session; i.e. setting up RFCOMM channel on L2CAP and starting RFCOMM multiplexing with the SABM command frame on DLCI 0 (zero)
RFCOMM Client	An RFCOMM client is an application that requests a connection to another application (RFCOMM server)
RFCOMM Server	An RFCOMM server is an application that awaits a connection from an RFCOMM client on another device. What happens after such a connection is established is not within the scope of this definition
RFCOMM Server Channel	This is a subfield of the TS 07.10 DLCI number. This abstraction is used to allow both server and client applications to reside on both sides of an RFCOMM session

Part F:2

IrDA INTEROPERABILITY



The IrOBEX protocol is utilized by the Bluetooth technology. In Bluetooth, OBEX offers the same features for applications as within the IrDA protocol hierarchy, enabling the applications to work over the Bluetooth protocol stack as well as the IrDA stack.

IrDA Interoperability

Bluetooth.

CONTENTS

1	Introduction	414
1.1	OBEX and Bluetooth Architecture.....	415
1.2	Bluetooth OBEX-Related Specifications	415
1.3	Other IrOBEX Implementations.....	416
2	OBEX Object and Protocol	417
2.1	Object.....	417
2.2	Session Protocol.....	417
2.2.1	Connect Operation	418
2.2.2	Disconnect Operation.....	419
2.2.3	Put Operation	419
2.2.4	Get Operation.....	420
2.2.5	Other Operations.....	420
3	OBEX over RFCOMM	421
3.1	OBEX Server Start-up on RFCOMM.....	421
3.2	Receiving OBEX Packets from Serial Port.....	421
3.3	Connection Establishment	422
3.4	Disconnection	422
3.5	Pushing and Pulling OBEX Packets over RFCOMM	422
4	OBEX over TCP/IP	423
4.1	OBEX Server Start-up on TCP/IP	423
4.2	Connection Establishment	423
4.3	Disconnection	424
4.4	Pushing and Pulling OBEX Packets over TCP	424
5	Bluetooth Application Profiles using OBEX.....	425
5.1	Synchronization	425
5.2	File Transfer	425
5.3	Object Push	426
6	References	427
7	List of Acronyms and Abbreviations.....	428

1 INTRODUCTION

The goal of this document is to enable the development of application programs that function well over both short-range RF and IR media. Each media type has its advantages and disadvantages but the goal is for applications to work over both. Rather than fragment the application domain, this document defines the intersection point where Bluetooth and IrDA applications may converge. That intersection point is IrOBEX [1].

IrOBEX is a session protocol defined by IrDA. This protocol is now also utilized by the Bluetooth technology, making it possible for applications to use either the Bluetooth radio technology or the IrDA IR technology. However, even though both IrDA and Bluetooth are designed for short-range wireless communications, they have some fundamental differences relating to the lower-layer protocols. IrOBEX will therefore be mapped over the lower layer protocols which are adopted by Bluetooth.

This document defines how IrOBEX (OBEX for short) is mapped over RFCOMM [2] and TCP/IP [3]. Originally, OBEX (Object Exchange Protocol) was developed to exchange data objects over an infrared link and was placed within the IrDA protocol hierarchy. However, it can appear above other transport layers, now RFCOMM and TCP/IP. At this moment, it is worth mentioning that the OBEX over TCP/IP implementation is an optional feature for Bluetooth devices supporting the OBEX protocol.

The IrOBEX specification [1] provides a model for representing objects and a session protocol, which structures the dialogue between two devices. The IrOBEX protocol follows a client/server **request-response** paradigm for the conversation format.

Bluetooth uses only the connection-oriented OBEX even though IrDA has specified the connectionless OBEX also. The reasons for the connection-oriented approach are:

- OBEX is mapped over the connection-oriented protocols in the Bluetooth architecture.
- Most of application profiles using OBEX and Bluetooth needs a connection-oriented OBEX to provide the functionality described for the features included in these profiles.
- The connectionless OBEX with the connection-oriented one would raise the interoperability problems, which are not desirable.

1.1 OBEX AND BLUETOOTH ARCHITECTURE

Figure 1.1 depicts part of the hierarchy of the Bluetooth architecture and shows the placement of the OBEX protocol and the application profiles using it (See also Section 5 on page 425). The protocols can also communicate with the service discovery DB even though the figure does not show it.

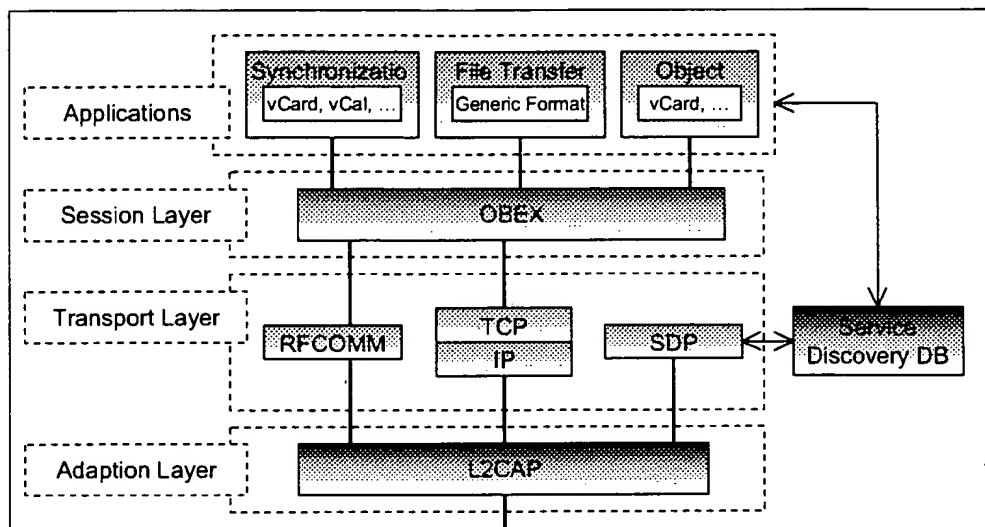


Figure 1.1: Part of Bluetooth Protocol Hierarchy

In the Bluetooth system, the purpose of the OBEX protocol is to enable the exchange of data objects. The typical example could be an object push of business cards to someone else. A more complex example is synchronizing calendars on multiple devices using OBEX. Also, the File Transfer applications can be implemented using OBEX. For the Object Push and Synchronization applications, content formats can be the vCard [4], vCalendar [5], vMessage [6], and vNotes [6] formats. The vCard, vCalendar, vMessage, and vNotes describe the formats for the electronic business card, the electronic calendaring and scheduling, the electronic message and mails, and the electronic notes, respectively.

1.2 BLUETOOTH OBEX-RELATED SPECIFICATIONS

Bluetooth Specification includes five separate specifications related to OBEX and applications using it:

1. Bluetooth IrDA Interoperability Specification (This specification)

- Defines how the applications can function over both Bluetooth and IrDA
- Specifies how OBEX is mapped over RFCOMM and TCP
- Defines the application profiles using OBEX over Bluetooth

2. Bluetooth Generic Object Exchange Profile Specification [7]

- Generic interoperability specification for the application profiles using OBEX
- Defines the interoperability requirements of the lower protocol layers (e.g. Baseband and LMP) for the application profiles

3. Bluetooth Synchronization Profile Specification [8]

- Application Profile for the Synchronization applications
- Defines the interoperability requirements for the applications within the Synchronization application profile
- Does not define the requirements for the Baseband, LMP, L2CAP, or RFCOMM.

4. Bluetooth File Transfer Profile Specification [9]

- Application Profile for the File Transfer applications
- Defines the interoperability requirements for the applications within the File Transfer application profile.
- Does not define the requirements for the Baseband, LMP, L2CAP, or RFCOMM.

5. Bluetooth Object Push Profile Specification [10]

- Application Profile for the Object Push applications
- Defines the interoperability requirements for the applications within the Object Push application profile.
- Does not define the requirements for the Baseband, LMP, L2CAP, or RFCOMM.

1.3 OTHER IROBEX IMPLEMENTATIONS

Over IR, OBEX has also been implemented over IrCOMM and Tiny TP. The Bluetooth technology does not define these protocols as transport protocols for OBEX, but they can be supported by independent software vendors if desired.

2 OBEX OBJECT AND PROTOCOL

This section is dedicated to the model of OBEX objects and the OBEX session protocol. The section is intended to be read with the IrOBEX specification [1].

2.1 OBJECT

The OBEX object model (Section 2 in [1]) describes how OBEX objects are presented. The OBEX protocol can transfer an object by using the **Put-** and **Get-**operations (See Section 2.2.3 and 2.2.4). One object can be exchanged in one or more **Put-**requests or **Get-**responses.

The model handles both information about the object (e.g. type) and object itself. It is composed of headers, which consist of a header ID and value (See Section 2.1 in [1]). The header ID describes what the header contains and how it is formatted, and the header value consists of one or more bytes in the format and meaning specified by Header ID. The specified headers are **Count**, **Name**, **Type**, **Length**, **Time**, **Description**, **Target**, **HTTP**, **Body**, **End of Body**, **Who**, **Connection ID**, **Application Parameters**, **Authenticate Challenge**, **Authenticate Response**, **Object Class**, and User-Defined Headers. These are explained in detail by Section 2.2 in the IrOBEX specification.

2.2 SESSION PROTOCOL

The OBEX operations are formed by **response-request** pairs. Requests are issued by the client and responses by the server. After sending a request, the client waits for a response from the server before issuing a new request. Each request packet consists of a one-byte opcode (See Section 3.3 in [1]), a two-byte length indicator, and required or optional data depending on the operation. Each response packet consists of a one-byte response code (See Section 3.2.1 in [1]), a two-byte length indicator, and required or optional data depending on the operation.

In the following subsections, the OBEX operations are explained in general.

2.2.1 Connect Operation

An OBEX session is started, when an application asks the first time to transmit an OBEX object. An OBEX client starts the establishment of an OBEX connection. The session is started by sending a **Connect**-request (See Section 3.3.1 in [1]). The request format is:

Byte 0	Bytes 1 and 2	Byte 3	Byte 4	Bytes 5 and 6	Byte 7 to n
0x80 (opcode)	Connect request packet length	OBEX version number	Flags	Maximum OBEX packet length	Optional headers

Note. The Big Endian format is used to define the byte ordering for the PDUs (requests and responses) in this specification as well as in the IrOBEX specification; i.e. the most significant byte (MSB) is always on left and the least significant byte (LSB) on right.

At the remote host, the **Connect**-request is received by the OBEX server, if it exists. The server accepts the connection by sending the successful response to the client. Sending any other response (i.e. a non-successful response) back to the client indicates a failure to make a connection. The response format is:

Byte 0	Bytes 1 and 2	Byte 3	Byte 4	Bytes 5 and 6	Byte 7 to n
Response code	Connect response packet length	OBEX version number	Flags	Maximum OBEX packet length	Optional headers

The response codes are listed in the Section 3.2.1 in the IrOBEX specification. The bytes 5 and 6 define the maximum OBEX packet length, which can be received by the server. This value may differ from the length, which can be received by the client. These **Connect**-request and response packets must each fit in a single packet.

Once a connection is established it remains 'alive', and is only disconnected by requests/responses or by failures (i.e. the connection is not automatically disconnected after each OBEX object has completely transmitted).

2.2.2 Disconnect Operation

The disconnection of an OBEX session occurs when an application, which is needed for an OBEX connection, is closed or the application wants to change the host to which the requests are issued. The client issues the **Disconnect**-request (See Section 3.3.2 in [1]) to the server. The request format is:

Byte 0	Bytes 1 and 2	Byte 3
0x81	Packet length	Optional headers

The request cannot be refused by the server. Thus, it has to send the response, and the response format is:

Byte 0	Bytes 1 and 2	Byte 3
0xA0	Response packet length	Optional response headers

2.2.3 Put Operation

When the connection has been established between the client and server the client is able to push OBEX objects to the server. The **Put**-request is used to push an OBEX object (See Section 3.3.3 in [1]). The request has the following format.

Byte 0	Bytes 1 and 2	Byte 3
0x02 (0x82 when Final bit set)	Packet length	Sequence of headers

A **Put**-request consists of one or more request packets, depending on how large the transferred object is, and how large the packet size is. A response packet from the server is required for every **Put**-request packet. Thus, one response is not permitted for several request packets, although they consist of one OBEX object. The response format is:

Byte 0	Bytes 1 and 2	Byte 3
Response code	Response packet length	Optional response headers

2.2.4 Get Operation

When the connection has been established between the client and server, the client is also able to pull OBEX objects from the server. The **Get**-request is used to pull an OBEX object (See Section 3.3.4 in [1]). The request has the following format.

Byte 0	Bytes 1 and 2	Byte 3
0x03 (0x83 when Final bit set)	Packet length	Sequence of headers starting with Name

The object is returned as a sequence of headers, and the client has to send a request packet for every response packet. The response format is:

Byte 0	Bytes 1 and 2	Byte 3
Response code	Response packet length	Optional response headers

2.2.5 Other Operations

Other OBEX operations consist of a **SetPath**-, and an **Abort**-operation. These are carefully explained in the Sections 3.3.5-6 in the IrOBEX specification. It is important to note that the client can send an **Abort**-request after each response – even in the middle of a request/response sequence. Thus, the whole OBEX object does not have to be received before sending an **Abort**-request. In addition to these operations, the IrOBEX specification facilitates user-defined operations, but their use may not necessarily be adopted in Bluetooth.

3 OBEX OVER RFCOMM

This section specifies how OBEX is mapped over RFCOMM, which is the multiplexing and transport protocol based on ETSI TS 07.10 [11] and which also provides a support for serial cable emulation. The Bluetooth devices supporting the OBEX protocol must satisfy the following requirements.

1. The device supporting OBEX must be able to function as either a client, a server, or both
2. All servers running simultaneously on a device must use separate RFCOMM server channels
3. Applications (service/server) using OBEX must be able to register the proper information into the service discovery database. This information for different application profiles is specified in the profile specifications

3.1 OBEX SERVER START-UP ON RFCOMM

When a client sends a connecting request, a server is assumed to be ready to receive requests. However, before the server is ready to receive (i.e. is running) certain prerequisites must be fulfilled before the server can enter the listening mode:

1. The server must open an RFCOMM server channel
2. The server must register its capabilities into the service discovery database

After this, other hosts are able to find the server if needed, and the server listens for get requests from clients.

3.2 RECEIVING OBEX PACKETS FROM SERIAL PORT

As discussed earlier, one object can be exchanged over one or more **Put**-requests or **Get**-responses (i.e. the object is received in one or several packets). However, if OBEX is running directly over the serial port, it does not receive packets from RFCOMM. Instead, a byte stream is received by OBEX from a serial port emulated by RFCOMM.

To detect packets in the byte stream, OBEX has to look for opcodes or response codes (See Chapter 2.2) depending on whether a packet is a request or a response. The opcodes and response code can be thought of as the start flags of packets. In OBEX packets, there is no 'end flag' that would indicate the end of a packet. However, after the opcode or response code, the length of a packet is received in the next two bytes. Thus, the whole length of a packet is known, and the boundary of two packets can be determined.

All data that is not recognized must be dumped. This could cause a synchronization problem but, considering the nature of the OBEX protocol, this is not a problem over RFCOMM, which provides reliable transport over Bluetooth.

3.3 CONNECTION ESTABLISHMENT

A client initiates the establishment of a connection. However, the following sequence of tasks must occur before the client is able to send the first request for data:

1. By using the SD protocol described in the SDP specification [12], the client must discover the proper information (e.g. RFCOMM channel) associated with the server on which the connection can be established
2. The client uses the discovered RFCOMM channel to establish the RFCOMM connection
3. The client sends the **Connect**-request to the server, to establish an OBEX session. The session is established correctly if the client receives a successful response from the server

3.4 DISCONNECTION

The disconnection of an OBEX session over RFCOMM is straightforward. The disconnection is done by using the **Disconnect**-request (See Section 2.2.2). When the client has received the response, the next operation is to close the RFCOMM channel assigned to the OBEX client.

3.5 PUSHING AND PULLING OBEX PACKETS OVER RFCOMM

Data is pushed in OBEX packets over RFCOMM by using **Put**-requests (See Section 2.2.3). After each request, a response is required before the next request with the data can be pushed.

Pulling data from a remote host happens by sending a **Get**-request (See Section 2.2.4). The data arrives in OBEX response packets. After each response, a new request has to be sent, to pull more data.

4 OBEX OVER TCP/IP

This section specifies how OBEX is mapped over the TCP/IP creating reliable connection-oriented services for OBEX. This specification does not define how TCP/IP is mapped over Bluetooth.

The Bluetooth devices, which support the OBEX protocol over TCP/IP, must satisfy the following requirements:

1. The device supporting OBEX must be able to function as either a client, or a server, or both
2. For the server, the TCP port number 650 is assigned by IANA. If an assigned number is not desirable, the port number can be a value above 1023. However, the use of the TCP port number (650) defined by IANA is highly recommended. The 0-1023 range is reserved by IANA (See [13])
3. The client must use a port number (on the client side), which is not within the 0-1023 range
4. Applications (service/server) using OBEX must be able to register the proper information into the service discovery database. This information for different application profiles is specified in the profile specifications

4.1 OBEX SERVER START-UP ON TCP/IP

When a client sends a **Put-** or **Get-**request, a server is assumed to be ready to receive requests. However, when the server is ready (i.e. is running), certain prerequisites must be fulfilled before the server can enter the listening mode:

1. The server must initialize a TCP port with the value 650 or value above 1023
2. The server registers its capabilities into the service discovery database

After this, other devices are able to find the server if needed, and the server listens for get requests from clients.

4.2 CONNECTION ESTABLISHMENT

A client initiates a connection. However, the following sequence of tasks must occur before a connection can be established:

1. By using, the SD protocol described in the SDP specification [12], the client discovers the proper information (e.g. TCP port number) associated with the server, to enable the connection can be established
2. The client initializes a socket associated to a TCP port number above 1023, and establishes a TCP connection with the host of the server
3. The client sends the **Connect-**request to the server, to establish an OBEX session. The session is established correctly if the client receives a successful response from the server.

4.3 DISCONNECTION

The disconnection of an OBEX session over TCP is straightforward. The disconnection is done by using the **Disconnect**-request (See Section 2.2.2). When the client has received the response, the next operation is to close the TCP port dedicated for this session.

4.4 PUSHING AND PULLING OBEX PACKETS OVER TCP

See Section 3.5.

5 BLUETOOTH APPLICATION PROFILES USING OBEX

Bluetooth SIG (Special Interest Group) has defined three separate application profiles using OBEX. These profiles are briefly introduced in this section.

5.1 SYNCHRONIZATION

Basically, the synchronization means comparing two object stores, determining their inequalities, and then unifying these two object stores. The Bluetooth devices supporting the synchronization may be desktop PCs, notebooks, PDAs, cellular phones, or smart phones.

The Bluetooth Synchronization profile uses the servers and clients compliant to the IrMC synchronization specified by IrDA (See Section 5 in [6]). The Bluetooth Synchronization servers and clients must support the level 4 synchronization functionality specified in the IrMC specification.

The actual logic of the synchronization engines which process the synchronization algorithm at the client device is implementation-specific. It is therefore left to the participating software vendors, and is not considered in the Bluetooth specifications.

The synchronization is not limited to one type of application. The Bluetooth synchronization (i.e. the IrMC synchronization) enables four different application classes:

1. Phone Book – provides a means for a user to manage contact records
2. Calendar – enables a user to manage calendar items, and can also be used for 'to-do' or task lists
3. Messaging – lets a user manage messages (e.g. e-mails)
4. Notes – provides a means for a user to manage small notes

The interoperability requirements for the Bluetooth Synchronization profile are defined in the Synchronization Profile [8] and Generic Object Exchange Profile [7] specifications.

5.2 FILE TRANSFER

At the minimum, the File Transfer profile is intended for sending and retrieving generic files to and from the Bluetooth device. The File Transfer service also facilitates the browsing of the remote Bluetooth device's folder.

The interoperability requirements for the Bluetooth File Transfer profile are defined in the File Transfer Profile [9] and Generic Object Exchange Profile [7] specifications.

5.3 OBJECT PUSH

The Object Push profile is the special case of the File Transfer Profile for beaming objects and optionally pulling the default objects. At a minimum, it offers the capability to exchange business cards, but is not limited to this service.

The interoperability requirements for the Object Push profile are defined in the Object Push Profile [10] and Generic Object Exchange Profile [7] specifications.

6 REFERENCES

- [1] Infrared Data Association, IrDA Object Exchange Protocol (IrOBEX), Version 1.2, April 1999
- [2] Bluetooth RFCOMM with TS 07.10, on page 385
- [3] Internet Engineering Task Force, IETF Directory List of RFCs (<http://www.ietf.org/rfc/>), May 1999.
- [4] The Internet Mail Consortium, vCard - The Electronic Business Card Exchange Format, Version 2.1, September 1996.
- [5] The Internet Mail Consortium, vCalendar - The Electronic Calendaring and Scheduling Exchange Format, Version 1.0, September 1996.
- [6] Infrared Data Association, IrMC (Ir Mobile Communications) Specification, Version 1.1, February 1999.
- [7] Bluetooth Generic Object Exchange Profile, see Volume 2.
- [8] Bluetooth Synchronization Profile, see Volume 2.
- [9] Bluetooth File Transfer Profile, see Volume 2.
- [10] Bluetooth Object Push Profile, see Volume 2.
- [11] ETSI, TS 07.10, Version 6.3.0
- [12] Bluetooth Service Discovery Protocol, see Volume 2.
- [13] Internet Assigned Numbers Authority, IANA Protocol/Number Assignments Directory (<http://www.iana.org/numbers.html>), May 1999.

7 LIST OF ACRONYMS AND ABBREVIATIONS

Abbreviation or Acronym	Meaning
GEOP	Generic Object Exchange Profile
IrDA	Infrared Data Association
IrMC	Ir Mobile Communications
L2CAP	Logical Link Control and Adaptation Protocol
LSB	Least Significant Byte
MSB	Most Significant Byte
OBEX	Object exchange protocol
PDU	Protocol Data Unit
RFCOMM	Serial cable emulation protocol based on ETSI TS 07.10
SD	Service Discovery
SDP	Service Discovery Protocol
SDDB	Service Discovery Database
TCP/IP	Transport Control Protocol/Internet Protocol

Part F:3

**TELEPHONY CONTROL
PROTOCOL SPECIFICATION**

TCS Binary

This document describes the Bluetooth Telephony Control protocol Specification – Binary (TCS *Binary*), using a bit-oriented protocol. This protocol defines the call control signalling for the establishment of speech and data calls between Bluetooth devices. In addition, it defines mobility management procedures for handling Bluetooth TCS devices.

CONTENTS

1	General Description	435
1.1	Overview	435
1.2	Operation between devices.....	435
1.3	Operation between layers	437
2	Call Control (CC)	439
2.1	Call States	439
2.2	Call Establishment	439
2.2.1	Call Request.....	439
2.2.2	Bearer selection	440
2.2.3	Overlap Sending.....	441
2.2.4	Call Proceeding	441
2.2.4.1	Call proceeding, enbloc sending.....	441
2.2.4.2	Call proceeding, overlap sending.....	442
2.2.4.3	Expiry of timer T310.....	442
2.2.5	Call Confirmation.....	442
2.2.6	Call Connection	442
2.2.7	Call Information	443
2.2.8	Non-selected user clearing.....	443
2.2.9	In-band tones and announcements.....	443
2.2.10	Failure of call establishment.....	444
2.2.11	Call Establishment Message Flow	445
2.3	Call Clearing.....	446
2.3.1	Normal Call Clearing	446
2.3.2	Abnormal Call Clearing	447
2.3.3	Clear Collision	447
2.3.4	Call Clearing Message Flow.....	448

3	Group Management (GM)	449
3.1	Overview	449
3.2	The Wireless User Group	449
3.2.1	Description	449
3.2.2	Encryption within the WUG	450
3.2.3	Unconscious pairing	450
3.3	Obtain Access Rights	451
3.3.1	Procedure description	451
3.3.2	Message flow	451
3.4	Configuration Distribution	452
3.4.1	Procedure Description	452
3.4.2	Message flow	452
3.5	Fast inter-member Access	453
3.5.1	Listen Request	453
3.5.2	Listen Accept	453
3.5.3	Listen Reject by the WUG Master	454
3.5.4	Listen Reject by the WUG Member	454
3.5.5	Message flow	454
4	Connectionless TCS (CL)	455
5	Supplementary Services (SS)	456
5.1	Calling Line Identity	456
5.2	DTMF start & stop	456
5.2.1	Start DTMF request	457
5.2.2	Start DTMF response	457
5.2.3	Stop DTMF request	457
5.2.4	Stop DTMF response	457
5.2.5	Message flow	457
5.3	Register Recall	458

6	Message formats	459
6.1	Call Control Message Formats	460
6.1.1	ALERTING	460
6.1.2	CALL PROCEEDING	460
6.1.3	CONNECT	461
6.1.4	CONNECT ACKNOWLEDGE	461
6.1.5	DISCONNECT	462
6.1.6	INFORMATION	462
6.1.7	PROGRESS	463
6.1.8	RELEASE	463
6.1.9	RELEASE COMPLETE	464
6.1.10	SETUP	464
6.1.11	SETUP ACKNOWLEDGE	465
6.1.12	Start DTMF	465
6.1.13	Start DTMF Acknowledge	466
6.1.14	Start DTMF Reject	466
6.1.15	Stop DTMF	466
6.1.16	Stop DTMF Acknowledge	467
6.2	Group Management Message Formats	467
6.2.1	ACCESS RIGHTS REQUEST	467
6.2.2	ACCESS RIGHTS ACCEPT	467
6.2.3	ACCESS RIGHTS REJECT	468
6.2.4	INFO SUGGEST	468
6.2.5	INFO ACCEPT	468
6.2.6	LISTEN REQUEST	469
6.2.7	LISTEN SUGGEST	469
6.2.8	LISTEN ACCEPT	469
6.2.9	LISTEN REJECT	470
6.3	TCS Connectionless Message Formats	470
6.3.1	CL INFO	470
7	Message coding	471
7.1	Overview	471
7.2	Protocol Discriminator	472
7.3	Message Type	472
7.4	Other Information Elements	474
7.4.1	Coding rules	474
7.4.2	Audio control	476
7.4.3	Bearer capability	476
7.4.4	Call class	479

7.4.5	Called party number.....	480
7.4.6	Calling party number.....	481
7.4.7	Cause.....	482
7.4.8	Clock offset	482
7.4.9	Company specific.....	483
7.4.10	Configuration data.....	484
7.4.11	Destination CID.....	485
7.4.12	Keypad facility	485
7.4.13	Progress indicator	485
7.4.14	SCO Handle.....	486
7.4.15	Sending complete	486
7.4.16	Signal	486
8	Message Error handling	487
8.1	Protocol Discrimination Error	487
8.2	Message Too Short or Unrecognized	487
8.3	Message Type or Message Sequence Errors.....	487
8.4	Information Element Errors.....	487
9	Protocol Parameters	489
9.1	Protocol Timers	489
10	References.....	490
11	List of Figures	491
12	List of Tables	492
	Appendix 1 - TCS Call States	493

1 GENERAL DESCRIPTION

1.1 OVERVIEW

The Bluetooth Telephony Control protocol Specification Binary (TCS *Binary*) is based on the ITU-T Recommendation Q.931[1], applying the symmetrical provisions as stated in Annex D of Q.931. The resulting text does not discriminate between user and network side, but merely between Outgoing Side (the party originating the call) and Incoming Side (the party terminating the call). Effort was made to only apply those changes necessary for Bluetooth and foreseen applications, enabling re-use of Q.931 to the largest extent possible.

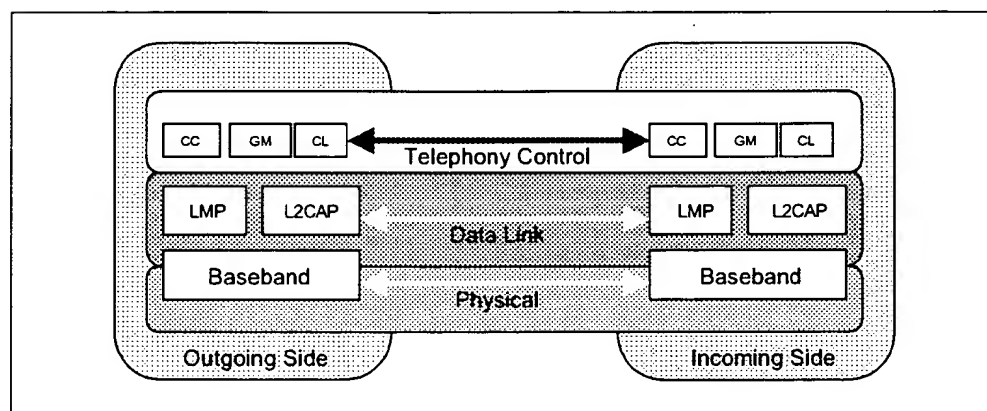


Figure 1.1: TCS within the Bluetooth stack

The TCS contains the following functionality:

- Call Control (CC) – signalling for the establishment and release of speech and data calls between Bluetooth devices
- Group Management – signalling to ease the handling of groups of Bluetooth devices
- ConnectionLess TCS (CL) – provisions to exchange signalling information not related to an ongoing call

1.2 OPERATION BETWEEN DEVICES

TCS uses point-to-point signalling and may use point-to-multipoint signalling. Point-to-point signalling is used when it is known to which side (Bluetooth device) a call needs to be established (*single-point configuration*).

Point-to-multipoint signalling may be used when there are more sides available for call establishment (*multi-point configuration*); e.g. when, for an incoming call, a home base station needs to alert all phones in range.

Point-to-point signalling is mapped towards a connection-oriented L2CAP channel, whereas point-to-multipoint signalling is mapped towards the connectionless L2CAP channel, which in turn is sent as broadcast information on the beacon channel (piconet broadcast).

Figure 1.2 illustrates point-to-point signalling to establish a voice or data call in a single-point configuration. First the other device is notified of the call request using the point-to-point signalling channel (A). Next, this signalling channel is used to further establish the speech or data channel (B).

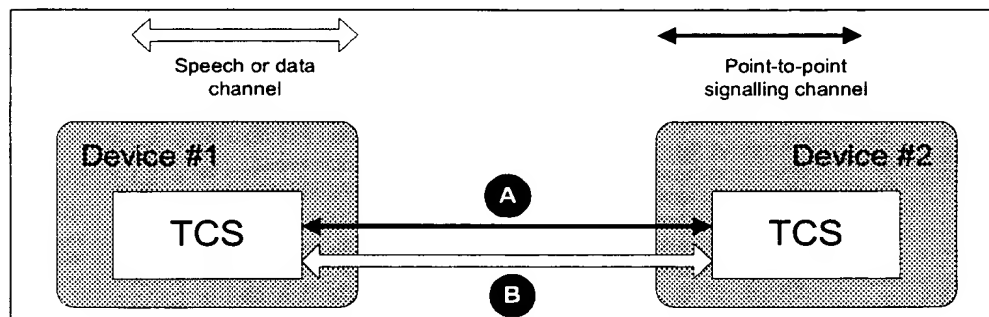


Figure 1.2: Point-to-point signalling in a single-point configuration

Figure 1.3 below illustrates how point-to-multipoint signalling and point-to-point signalling is used to establish a voice or data call in a multi-point configuration. First all devices are notified of the call request using point-to-multipoint signalling channel (A). Next, one of the devices answers the call on the point-to-point signalling channel (B); this signalling channel is used to further establish the speech or data channel (C).

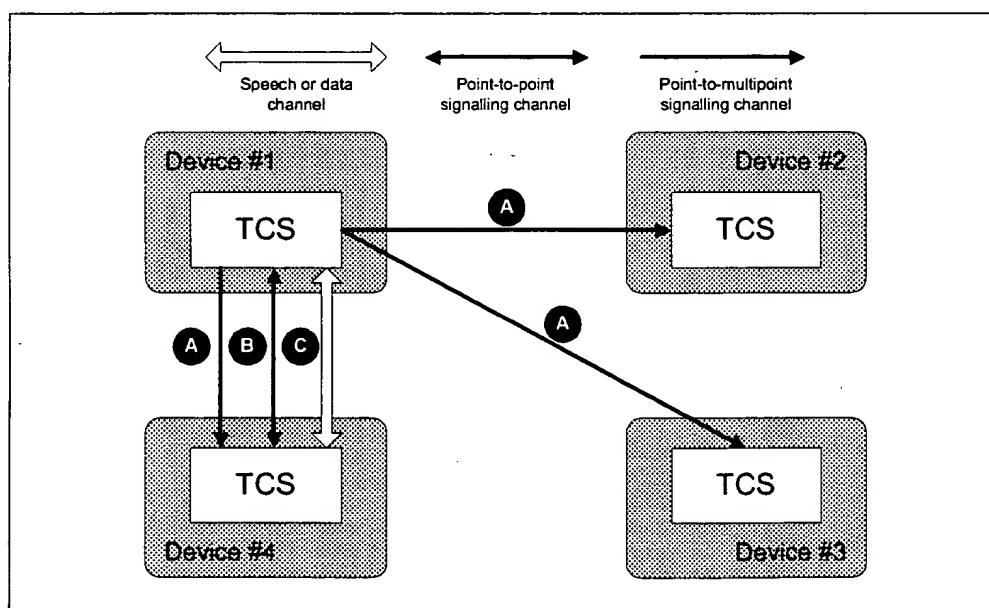


Figure 1.3: Signalling in a multi-point configuration

1.3 OPERATION BETWEEN LAYERS

TCS implementations should follow the general architecture described below (note that, for simplicity, handling of data calls is not drawn).

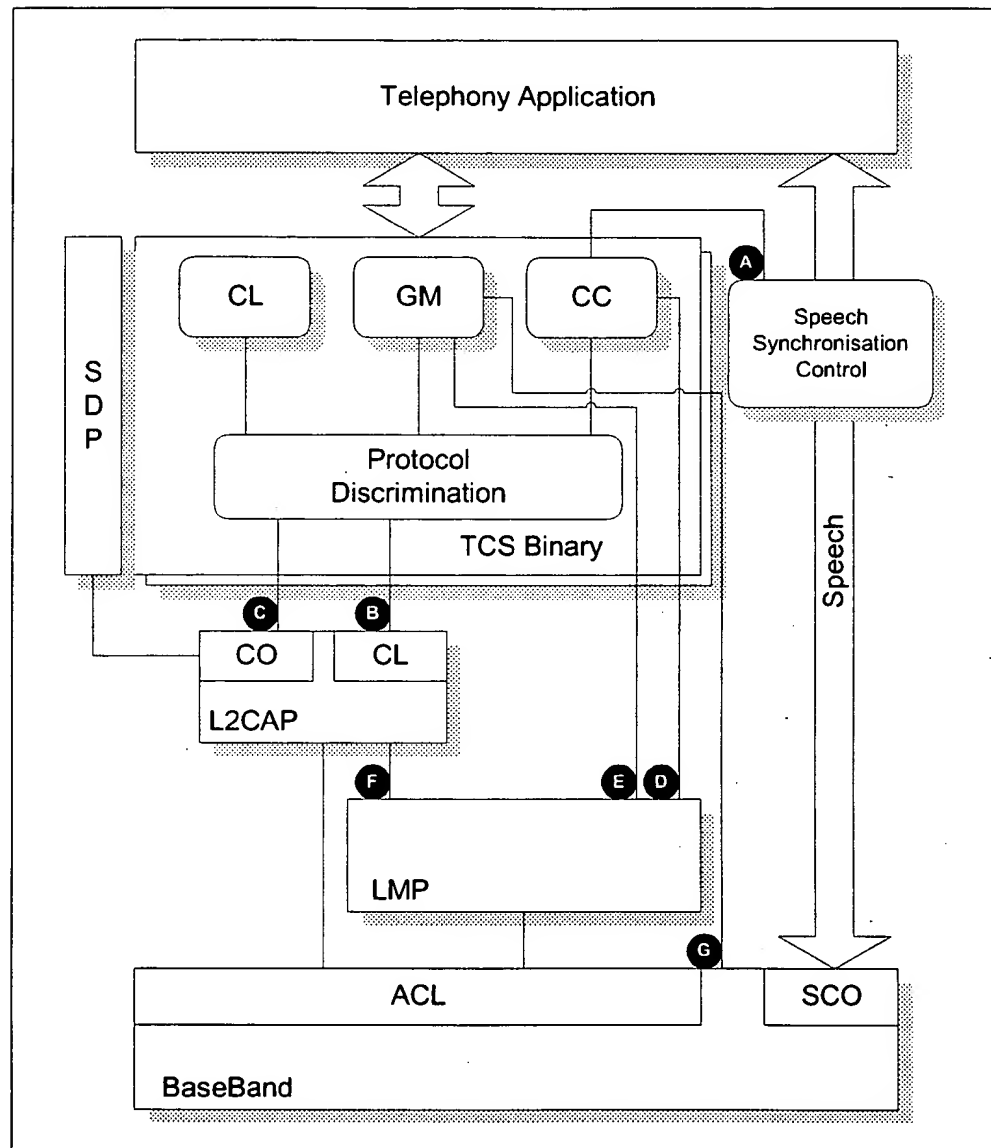


Figure 1.4: TCS Architecture

The internal structure of TCS Binary contains the functional entities Call Control, Group Management and ConnectionLess as described in Section 1.1 on page 435, complemented with the Protocol Discrimination which, based upon the TCS internal protocol discriminator, routes traffic to the appropriate functional entity.

To handle more calls simultaneously, multiple instances of TCS Binary may exist at the same time. Discrimination between the multiple instances can be based on the L2CAP channel identifier.

TCS Binary interfaces with a number of other (Bluetooth) entities to provide its (telephone) services to the application. The interfaces are identified in Figure 1.4 above, and information is exchanged across these interfaces for the following purposes:

- A The Call Control entity provides information to the speech synchronization control about when to connect (disconnect) the speech paths. This information is based upon the call control messages (e.g. reception of CONNECT ACKNOWLEDGE or DISCONNECT, see Section 2 on page 439)
- B To send a SETUP message (see Section 2.2.1 on page 439) using point-to-multipoint signalling, it is delivered on this interface to L2CAP for transmission on the connectionless channel. The other way round – L2CAP uses this interface to inform TCS of a SETUP message received on the connectionless channel. The connectionless L2CAP channel maps onto the piconet broadcast
- C Whenever a TCS message needs to be sent using point-to-point signalling, it is delivered on this interface to L2CAP for transmission on a connection-oriented channel. During L2CAP channel establishment specific quality of service to be used for the connection will be indicated, in particular the usage of low power modes (L2CAP will inform LMP about this – interface F)
- D The Call Control entity controls the LMP directly, for the purpose of establishing and releasing SCO links
- E & G. The Group Management entity controls the LMP and LC/Baseband directly during initialization procedures to control (for example) the inquiry, paging and pairing.

2 CALL CONTROL (CC)

2.1 CALL STATES

The call states used by the TCS are those identified in Q.931[1], for the user side only. To allow for implementation within computing power- and memory-restricted devices, only a subset of the states is mandatory for TCS based implementations. This mandatory subset is termed **Lean TCS**.

The states are named as follows. States in bold are mandatory states, part of Lean TCS:

General States

Null (0)

Active (10)

Disconnect request (11)

Disconnect indication (12)

Release request (19)

Outgoing Side States

Call initiated (1)

Overlap sending (2)

Outgoing call proceeding (3)

Call delivered (4)

Incoming Side States

Call present (6)

Call received (7)

Connect request (8)

Incoming call proceeding (9)

Overlap receiving (25)

These states, together with the state transitions, have been indicated in the state diagram contained in Appendix 1 – TCS Call States. For clarity, a separate state diagram has been included for Lean TCS.

2.2 CALL ESTABLISHMENT

A connection-oriented L2CAP channel between the Outgoing and Incoming Side shall be available before any of the CC procedures can operate.

Additionally, in a multi-point configuration (see Section 1.2 on page 435), a connectionless L2CAP channel shall be available between the Outgoing and Incoming Side.

2.2.1 Call Request

The Outgoing Side initiates call establishment by sending a SETUP message, and starting timer T303.

In case of a single-point configuration (see Section 1.2 on page 435), the SETUP message is delivered on the connection-oriented channel.

In case of a multi-point configuration (see Section 1.2 on page 435), the SETUP message may be delivered on the connection-less channel. This causes the SETUP message to be transmitted as a broadcast message at every beacon instant (as described in Baseband Specification Section 10.8.4 on page 115).

If no response (as prescribed in Section 2.2.4 on page 441) is received from the Incoming Side before timer T303 expires, the Outgoing Side shall:

1. If the SETUP message was delivered on a connection-less channel, return to the Null state. This stops the transmission of the SETUP message.
2. If the SETUP message was delivered on a connection-oriented channel, send a RELEASE COMPLETE message to the Incoming Side. This message should contain cause # 102, *recovery on timer expiry*.

The SETUP message shall always contain the call class. It shall also contain all the information required by the Incoming Side to process the call. The number digits within the Called party number information element may optionally be incomplete, thus requiring the use of overlap sending (Section 2.2.3 on page 441). The SETUP message may optionally contain the Sending complete information element in order to indicate that the number is complete.

Following the transmission of the SETUP message, the Outgoing Side shall enter the Call initiated state. On receipt of the SETUP message the Incoming Side shall enter the Call present state.

2.2.2 Bearer selection

The SETUP message sent during the Call Request may contain the Bearer capability information element, to indicate the requested bearer. The Incoming Side may negotiate on the requested bearer by including a Bearer capability information element in the first message in response to the SETUP message.

The Bearer capability information element indicates which lower layer resources (the *bearer channel*) are used during a call. If bearer capability 'Synchronous Connection-Oriented (SCO)' is indicated, an SCO link will be used, with the indicated packet type and voice coding to enable speech calls. If bearer capability 'Asynchronous Connection-Less (ACL)' is indicated, an ACL link will be used. On top of this, there will be an L2CAP channel with indicated QoS requirements, to enable data calls. If bearer capability 'None' is indicated, no separate bearer channel will be established.

Note: it is the responsibility of the implementation to assure that the bearer capability as indicated is available to the call.

2.2.3 Overlap Sending

If the received SETUP message does not contain a Sending complete indication information element, and contains either –

- a) incomplete called-number information, or
- b) called-number information which the Incoming Side cannot determine to be complete,

then the Incoming Side shall start timer T302, send a SETUP ACKNOWLEDGE message to the Outgoing Side, and enter the Overlap receiving state.

When the SETUP ACKNOWLEDGE message is received, the Outgoing Side shall enter the Overlap sending state, stop timer T303, and start timer T304.

After receiving the SETUP ACKNOWLEDGE message, the Outgoing Side shall send the remainder of the call information (if any) in the called party number information element of one or more INFORMATION messages.

The Outgoing Side shall restart timer T304 when each INFORMATION message is sent.

The INFORMATION message, which completes the information sending, may contain a sending complete information element. The Incoming Side shall restart timer T302 on receipt of every INFORMATION message not containing a sending complete indication, if it cannot determine that the called party number is complete.

At the expiry of timer T304, the Outgoing Side shall initiate call clearing in accordance with Section 2.3.1 with cause #102, *recovery on timer expiry*.

At the expiry of timer T302, the Incoming Side shall:

- if it determines that the call information is incomplete, initiate call clearing in accordance with Section 2.3.1 with cause #28, *invalid number format*.
- otherwise the Incoming Side shall reply with a CALL PROCEEDING, ALERTING or CONNECT message.

2.2.4 Call Proceeding

2.2.4.1 Call proceeding, enbloc sending

If enbloc sending is used (i.e. the Incoming Side can determine it has received sufficient information in the SETUP message from the Outgoing Side to establish the call) the Incoming Side shall send a CALL PROCEEDING message to the Outgoing Side to acknowledge the SETUP message and to indicate that the call is being processed. Upon receipt of the CALL PROCEEDING message, the Outgoing Side shall enter the Outgoing Call proceeding state stop

timer T303 and start timer T310. After sending the CALL PROCEEDING message, the Incoming Side shall enter the Incoming Call proceeding state.

2.2.4.2 Call proceeding, overlap sending

Following the occurrence of one of these conditions –

- the receipt by the Incoming Side of a Sending complete indication, or
- analysis by the Incoming Side that all call information necessary to effect call establishment has been received,

the Incoming Side shall send a CALL PROCEEDING message to the Outgoing Side, stop timer T302, and enter the Incoming Call proceeding state.

When the Outgoing Side receives of the CALL PROCEEDING message it shall enter the Outgoing Call proceeding state, stop timer T304 and, if applicable, start timer T310.

2.2.4.3 Expiry of timer T310

On expiry of T310 (i.e. if the Outgoing Side does not receive an ALERTING, CONNECT, DISCONNECT or PROGRESS message), the Outgoing Side shall initiate call clearing in accordance with Section 2.3.1 on page 446 with cause #102, *recovery on timer expiry*.

2.2.5 Call Confirmation

Upon receiving an indication that user alerting has been initiated at the called address, the Incoming Side shall send an ALERTING message, and shall enter the Call received state.

When the Outgoing Side receives the ALERTING message, the Outgoing Side may begin an internally generated alerting indication and shall enter the Call delivered state. The Outgoing Side shall stop timer T304 (in case of overlap receiving), stop timer T303 or T310 (if running), and start timer T301 (unless another internal altering supervision timer function exists).

On expiry of T301, the Outgoing Side shall initiate call clearing in accordance with Section 2.3.1 on page 446 with cause #102, *recovery on timer expiry*.

2.2.6 Call Connection

An Incoming Side indicates acceptance of an incoming call by sending a CONNECT message to the Outgoing Side, and stopping the user alerting. Upon sending the CONNECT message the Incoming Side shall start timer T313.

On receipt of the CONNECT message, the Outgoing Side shall stop any internally generated alerting indications, shall stop (if running) timers T301, T303, T304, and T310, shall complete the requested bearer channel to the Incoming Side, shall send a CONNECT ACKNOWLEDGE message, and shall enter the Active state.

The CONNECT ACKNOWLEDGE message indicates completion of the requested bearer channel. Upon receipt of the CONNECT ACKNOWLEDGE message, the Incoming Side shall connect to the bearer channel, stop timer T313 and enter the Active state.

When timer T313 expires prior to the receipt of a CONNECT ACKNOWLEDGE message, the Incoming Side shall initiate call clearing in accordance with Section 2.3.1 on page 446 with cause #102, *recovery on timer expiry*.

2.2.7 Call Information

While in the Active state, both sides may exchange any information related to the ongoing call using INFORMATION messages.

2.2.8 Non-selected user clearing

When the call has been delivered on a connection-less channel (in case of a multi-point configuration), in addition to sending a CONNECT ACKNOWLEDGE message to the Incoming Side selected for the call, the Outgoing Side shall send a RELEASE message (indicating cause #26, *non-selected user clearing*) to all other Incoming Sides that have sent SETUP ACKNOWLEDGE, CALL PROCEEDING, ALERTING, or CONNECT messages in response to the SETUP message. These RELEASE messages are used to notify the Incoming Sides that the call is no longer offered to them.

2.2.9 In-band tones and announcements

When the Incoming Side provides in-band tones/announcements, and if the requested bearer implies speech call, the Incoming Side will first complete the bearer channel (if not already available). Then a progress indicator #8, *in-band information or appropriate pattern is now available* is sent simultaneously with the application of the in-band tone/announcement. This progress indicator may be included in any call control message that is allowed to contain the progress indicator information element or, if there is no call state change, in a dedicated PROGRESS message.

Upon receipt of this message, the Outgoing Side may connect (if not already connected) to the bearer channel to receive the in-band tone/announcement.

2.2.10 Failure of call establishment

In the Call present, Overlap receiving, Incoming call proceeding, or Call received states, the Incoming Side may initiate clearing as described in Section 2.3 on page 446 with a cause value indicated. Examples of some the cause values that may be used to clear the call, when the Incoming Side is in the Call present, Overlap receiving, or Incoming call proceeding state are the following:

- #1 unassigned (unallocated) number*
- #3 no route to destination*
- #17 user busy*
- #18 no user responding*
- #22 number changed*
- #28 invalid number format (incomplete number)*
- #34 no circuit/channel available*
- #44 requested circuit/channel not available*
- #58 bearer capability not presently available*
- #65 bearer capability not implemented*

Examples of two of the cause values that may be used to clear the call when the Incoming Side is in the Call received state are as follows:

- #19 no answer from user (user alerted)*
- #21 call rejected by user*

2.2.11 Call Establishment Message Flow

The figure below provides a complete view of the messages exchanged during successful Call Establishment, as described in the sections above. The mandatory messages, part of the Lean TCS, are indicated by a solid arrow. A dotted arrow indicates the optional messages. A triangle indicates a running timer.

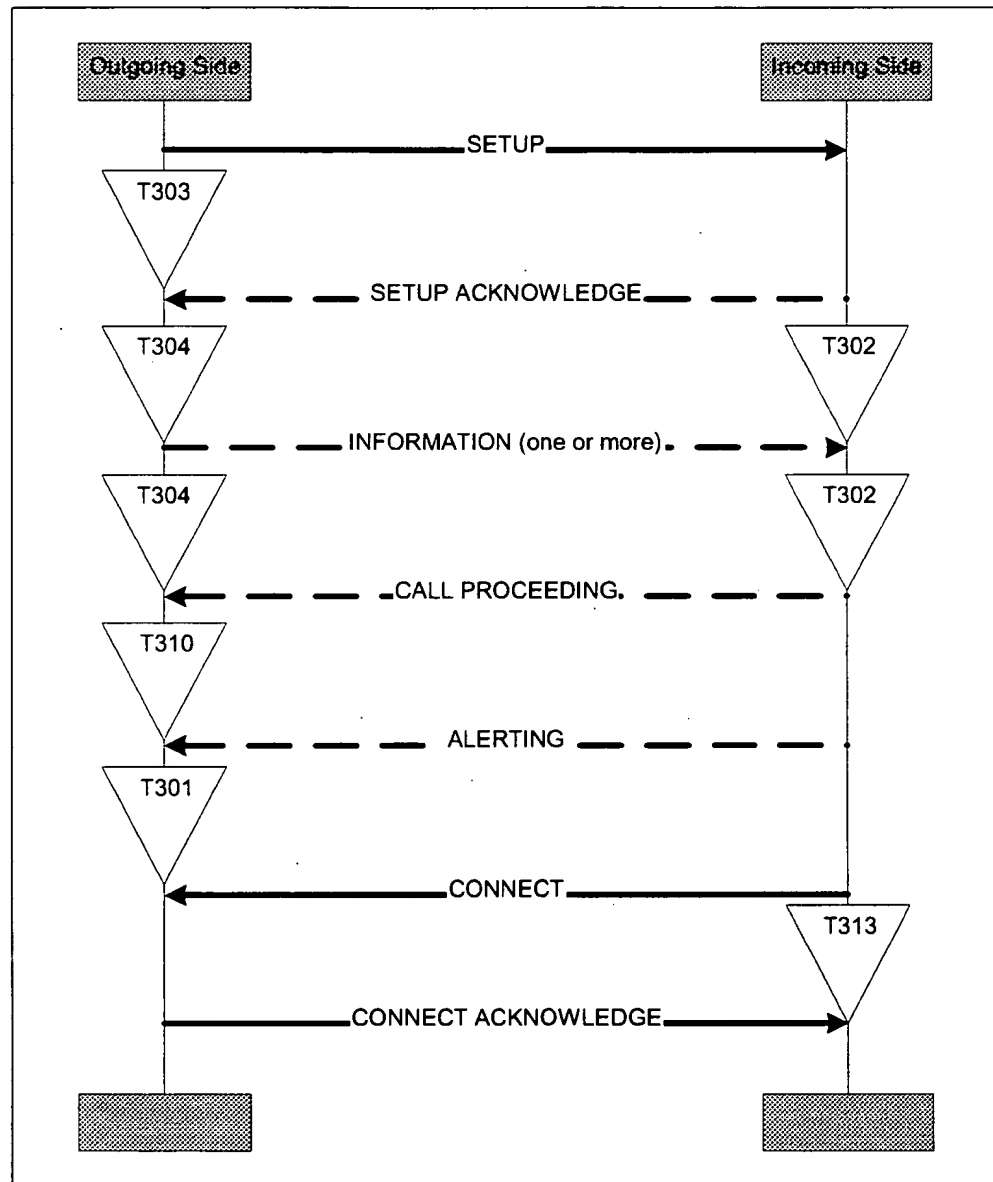


Figure 2.1: Call establishment message flow

2.3 CALL CLEARING

2.3.1 Normal Call Clearing

Apart from the exceptions identified in Section 2.3.2 on page 447, the clearing procedures are symmetrical and may be initiated by either the Outgoing or the Incoming Side. In the interest of clarity, the following procedures describe only the case where the Outgoing Side initiates clearing.

On sending or receiving any call clearing message, any protocol timer other than T305 and T308 shall be stopped.

The Outgoing Side shall initiate clearing by sending a DISCONNECT message, starting timer T305, disconnecting from the bearer channel, and entering the Disconnect request state.

The Incoming Side shall enter the Disconnect indication state upon receipt of a DISCONNECT message. This message prompts the Incoming Side to disconnect from the bearer channel. Once the channel used for the call has been disconnected, the Incoming Side shall send a RELEASE message to the Outgoing Side, start timer T308, and enter the Release request state.

On receipt of the RELEASE message the Outgoing Side shall cancel timer T305, release the bearer channel, send a RELEASE COMPLETE message, and return to the Null state.

Following the receipt of a RELEASE COMPLETE message from the Outgoing Side, the Incoming Side shall stop timer T308, release the bearer channel, and return to the Null state.

If the Outgoing Side does not receive a RELEASE message in response to the DISCONNECT message before timer T305 expires, it shall send a RELEASE message to the Incoming Side with the cause number originally contained in the DISCONNECT message, start timer T308 and enter the Release request state.

If in the Release request state, a RELEASE COMPLETE message is not received before timer T308 expires, the side that expected the message shall return to the Null state.

Clearing by the called user employing user-provided tones/announcements

In addition to the procedures described above, if the requested bearer signals a speech call, the Outgoing Side may apply in-band tones/announcements in the clearing phase. When in-band tones/announcements are provided, the Outgoing Side will first complete the bearer channel (if not already available), and next send the DISCONNECT message containing progress indicator #8, *in-band information or appropriate pattern is now available*.

Upon receipt of this message, the Incoming Side may connect (if not already connected) to the bearer channel to receive the in-band tone/announcement, and enter the Disconnect indication state.

The Incoming Side may subsequently continue clearing (before the receipt of a RELEASE from the Outgoing Side) by disconnecting from the bearer channel, sending a RELEASE message, starting timer T308, and entering the Release request state.

2.3.2 Abnormal Call Clearing

Under normal conditions, call clearing is initiated when either side sends a DISCONNECT message and follows the procedures defined in Section 2.3.1 on page 446. The only exceptions to the above rule are as follows:

- a In response to a SETUP message, the Incoming Side can reject a call (e.g. because of unavailability of suitable resources) by responding with a RELEASE COMPLETE message provided no other response has previously been sent, and enter the Null state
- b In case of a multi-point configuration, non-selected user call clearing will be initiated with RELEASE message(s) from the Outgoing Side (Section 2.2.8 on page 443)
- c In case of a multi-point configuration, where the SETUP message is delivered on an connection-less channel, if a remote (calling) user disconnect indication is received during call establishment, any Incoming Side which has responded, or subsequently responds, shall be cleared by a RELEASE message, and the procedures of Section 2.3.1 on page 446 are then followed for that user. The Outgoing Side enters the Null state upon completion of clearing procedures for all responding Incoming Sides.

2.3.3 Clear Collision

Clear collision occurs when the Incoming and the Outgoing Sides simultaneously transfer DISCONNECT messages. When either side receives a DISCONNECT message while in the Disconnect request state, the side shall stop timer T305, disconnect the bearer channel (if not disconnected), send a RELEASE message, start timer T308, and enter the Release request state.

Clear collision can also occur when both sides simultaneously transfer RELEASE messages. The entity receiving such a RELEASE message while within the Release request state shall stop timer T308, release the bearer channel, and enter the Null state (without sending or receiving a RELEASE COMPLETE message).

2.3.4 Call Clearing Message Flow

The figure below provides the complete view on the messages exchanged during normal Call Clearing, as described in the sections above. All messages are mandatory.

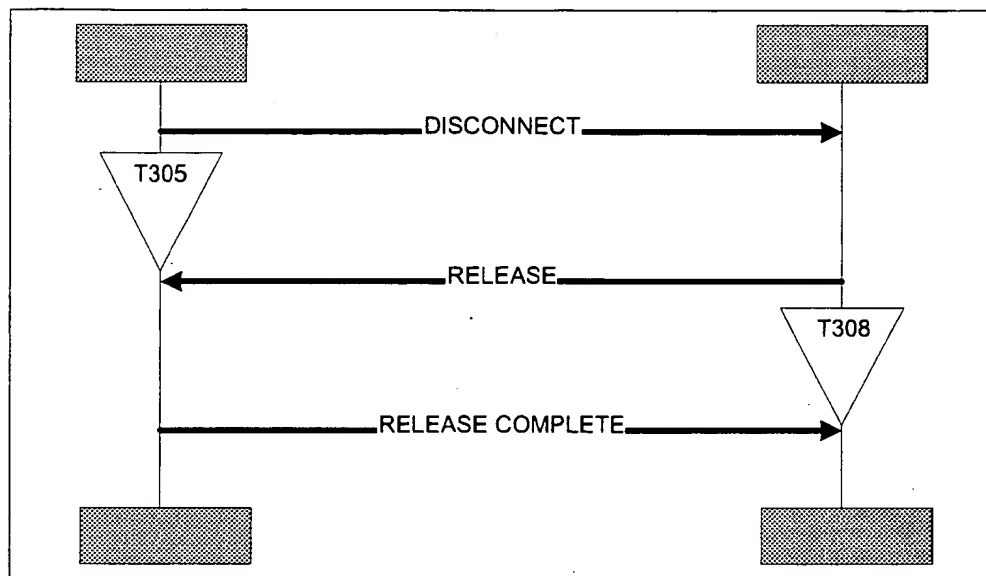


Figure 2.2: Call clearing message flow

3 GROUP MANAGEMENT (GM)

3.1 OVERVIEW

The Group Management entity provides procedures for managing a group of devices.

The following procedures are supported:

- Obtain access rights (Section 3.3 on page 451)
enables the requesting device to use the telephony services of another device, part of a group of devices
- Configuration distribution (Section 3.4 on page 452)
facilitates the handling and operation of a group of devices
- Fast inter-member access (Section 3.5 on page 453)
enables faster contact establishment between devices of the same group

A connection-oriented L2CAP channel between devices shall be available before any of the GM procedures can operate.

For group management, the concept of Wireless User Group (WUG) is used.

3.2 THE WIRELESS USER GROUP

3.2.1 Description

A WUG consists of a number of Bluetooth units supporting TCS. One of the devices is called the WUG master. The WUG master is typically a gateway, providing the other Bluetooth devices – called WUG members – with access to an external network. All members of the WUG in range are members of a piconet (active or parked). Master of this piconet is always the WUG master.

The main relational characteristics of a WUG are:

- All units that are part of a WUG know which unit is the WUG master and which other units are member of this WUG. WUG members receive this information from the WUG master.
- When a new unit has paired with the WUG master, it is able to communicate and perform authentication and encryption with any other unit part of the WUG without any further pairing/initialization. The WUG master provides the required authentication and encryption parameters to the WUG members.

Both relational characteristics are maintained through the Configuration distribution procedure.

3.2.2 Encryption within the WUG

In order to allow for encrypted transmission on the connectionless L2CAP channel, the WUG master issues a temporary key (K_{master}). As a Bluetooth unit is not capable of switching between two or more encryption keys in real time, this key is normally also used for encrypted transmission on the connection-oriented channel (individually addressed traffic). Since the WUG master piconet may be in operation for extended periods without interruption, the K_{master} shall be changed periodically.

In order to allow for authentication and encryption to be performed between WUG members, the WUG master may use the Configuration distribution procedure to issue link keys that the WUG members use for communication with each other. Just as if pairing had created these keys, the keys are unique to a pair of WUG members and hence a WUG member uses a different key for every other WUG member it connects to.

The Configuration distribution shall always be performed using encrypted links. The K_{master} shall not be used for encryption; rather the WUG master shall ensure that the semi-permanent key for the specific WUG member addressed shall be used.

3.2.3 Unconscious pairing

For TCS, pairing a device with the WUG master implies pairing a device with all members of the WUG. This is achieved using the Configuration distribution procedure. This avoids the user of the device having to pair with each and every device of the WUG individually.

In Bluetooth, pairing is not related to a specific service but rather to a specific device. After pairing, all services provided by a device are accessible, if no further application- or device-specific protection is provided.

Without further provisions, pairing a device with the WUG master implies that all services provided by the new device are accessible to all other WUG members. And vice versa, without further provisions, the new device can access all services provided by other WUG members.

For this reason, implementers of TCS – and in particular the Configuration distribution procedure – are recommended to add provisions where:

1. a new device entering the WUG is not mandated to initiate the Obtain access rights procedure to become a WUG member, and is consequently only able to use the services provided by the WUG master (gateway)
2. a WUG master can reject a request to obtain access rights
3. a WUG member is not forced to accept the pairing information received during the Configuration distribution

This applies in particular to devices offering more than just TCS- related services.

3.3 OBTAIN ACCESS RIGHTS

Using the Obtain access rights procedure, a device can obtain the rights to use the telephony services provided by another device, part of a WUG.

3.3.1 Procedure description

A device requests access rights by sending an ACCESS RIGHTS REQUEST message and starting timer T401. Upon receipt of the ACCESS RIGHTS REQUEST message, the receiving device accepts the request for access rights by sending an ACCESS RIGHTS ACCEPT.

When the requesting device receives the ACCESS RIGHTS ACCEPT, it shall stop timer T401. Then, the access rights procedure has completed successfully.

If no response has been received before the expiration of timer T401, the requesting device shall consider the request for access rights to be denied.

If, upon receipt of the ACCESS RIGHTS REQUEST message, the receiving device is for some reason unable to accept the access rights, it shall reply with an ACCESS RIGHTS REJECT message. Upon receipt of an ACCESS RIGHTS REJECT message, the requesting device shall stop timer T401 and consider the request for access rights to be denied.

3.3.2 Message flow

The figure below provides the complete view on the messages exchanged during the Obtain access rights procedure.

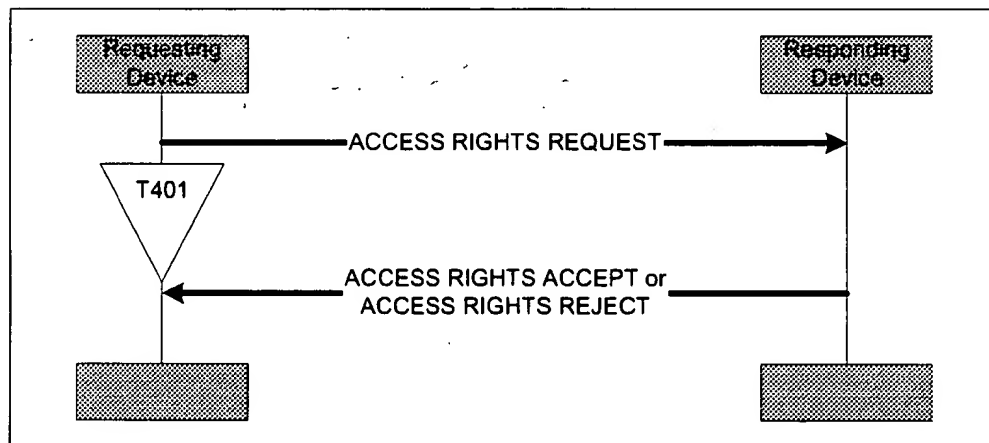


Figure 3.1: Obtain access rights message flow

3.4 CONFIGURATION DISTRIBUTION

The units in the WUG need to be informed about changes in the WUG; e.g. when a unit is added or removed. The Configuration distribution procedure is used to exchange this data.

When a WUG configuration change occurs, the WUG master initiates the Configuration distribution procedure on all WUG members. The WUG master keeps track of which WUG members have been informed of WUG configuration changes.

Some WUG members may be out of range and may therefore not be reached. The update of these WUG members will be performed when these members renew contact with the WUG master.

3.4.1 Procedure Description

The WUG master initiates the Configuration distribution procedure by starting timer T403, and transferring the INFO SUGGEST message. The INFO SUGGEST message contains the complete WUG configuration information. Upon receipt of the INFO SUGGEST message, the WUG member shall send an INFO ACCEPT message, to acknowledge the proper receipt of the WUG configuration information.

When the WUG master receives the INFO ACCEPT, the timer T401 is stopped, and the Configuration distribution procedure has completed successfully. On expiry of timer T403, the Configuration distribution procedure is terminated.

3.4.2 Message flow

The figure below provides the complete view on the messages exchanged during the Configuration distribution procedure, as described in the sections above.

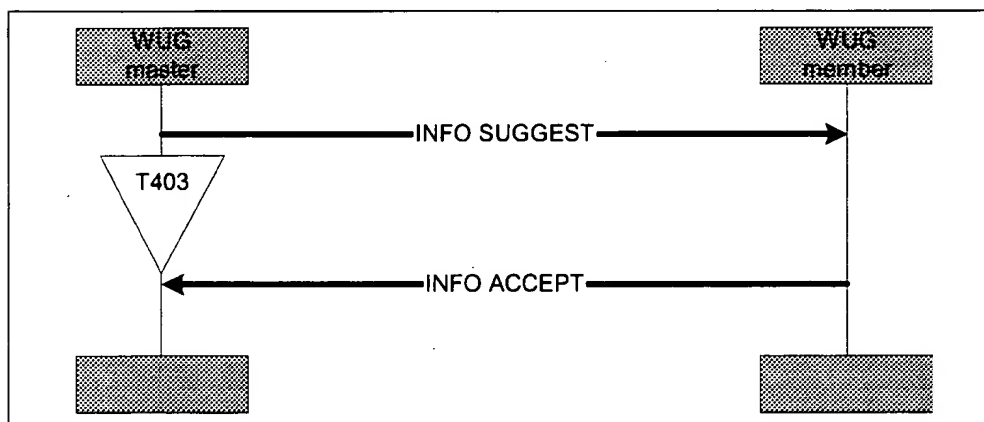


Figure 3.2: Configuration distribution message flow

3.5 FAST INTER-MEMBER ACCESS

When two WUG members are both active in the WUG master piconet, a WUG member can use the Fast inter-member access procedure to obtain fast access to another WUG member. With the Fast inter-member access procedure, the originating WUG member obtains clock information from the terminating WUG member and forces the terminating WUG member to go into PAGE_SCAN for a defined period (T406).

3.5.1 Listen Request

The originating WUG member initiates the Fast inter-member access procedure by starting timer T404 and transferring the LISTEN REQUEST message to the WUG master, indicating the WUG member with which it wishes to establish contact.

If, before expiry of timer T404, the originating WUG member receives no response to the LISTEN REQUEST message, the originating WUG member shall terminate the procedure.

3.5.2 Listen Accept

Upon receipt of the LISTEN REQUEST message, the WUG master checks that the indicated WUG member is part of the WUG. If this is the case, the WUG master initiates the Fast inter-member access towards the terminating WUG member side by starting timer T405 and sending the LISTEN SUGGEST message to the terminating WUG member.

Upon receipt of the LISTEN SUGGEST message, the terminating WUG member confirms the suggested action (internal call) by sending a LISTEN ACCEPT message to the WUG master. This message contains the terminating WUG member's clock offset. After sending the LISTEN ACCEPT, the terminating WUG member shall go to PAGE-SCAN state, for T406 seconds, to enable connection establishment by the originating WUG member.

Upon receipt of the LISTEN ACCEPT message, the WUG master stops timer T405, and informs the originating WUG member of the result of the WUG fast inter-member access by sending a LISTEN ACCEPT message. This message contains the terminating WUG member's clock offset. Upon receipt of the LISTEN ACCEPT message, the originating WUG member stops timer T404, and starts paging the terminating WUG member.

If no response to the LISTEN SUGGEST message is received by the WUG master before the first expiry of timer T405, then the WUG master shall terminate the Fast inter-member access procedure by sending a LISTEN REJECT message to both originating and terminating WUG member using cause #102, *recovery on timer expiry*.

3.5.3 Listen Reject by the WUG Master

If the WUG master rejects the Fast inter-member access procedure, it sends a LISTEN REJECT message to the originating WUG member.

Valid cause values are:

- #1, *Unallocated (unassigned) number* (when the indicated WUG member is not part of the WUG)
- #17, *User busy* (in case terminating WUG member is engaged in an external call)
- #20, *Subscriber absent* (upon failure to establish contact with the terminating WUG member), or
- any cause value indicated in a LISTEN REJECT message received from/sent to the terminating WUG member.

Upon receipt of the LISTEN REJECT message, the originating WUG member stops timer T404, and terminates the procedure.

3.5.4 Listen Reject by the WUG Member

If the terminating WUG member rejects the suggested action received in the LISTEN SUGGEST message, it sends a LISTEN REJECT message to the WUG master. Valid cause value is #17, *User busy* (in case terminating WUG member is engaged in another internal call).

Upon receipt of the LISTEN REJECT, the WUG master stops timer T405, and continues as described in Section 3.5.3 on page 454.

3.5.5 Message flow

The figure below provides a view of the messages exchanged during Fast inter-member access, as described in the sections above. A successful Fast inter-member access procedure ends with the terminating WUG member going into page scan, thus allowing the originating WUG member to contact him directly.

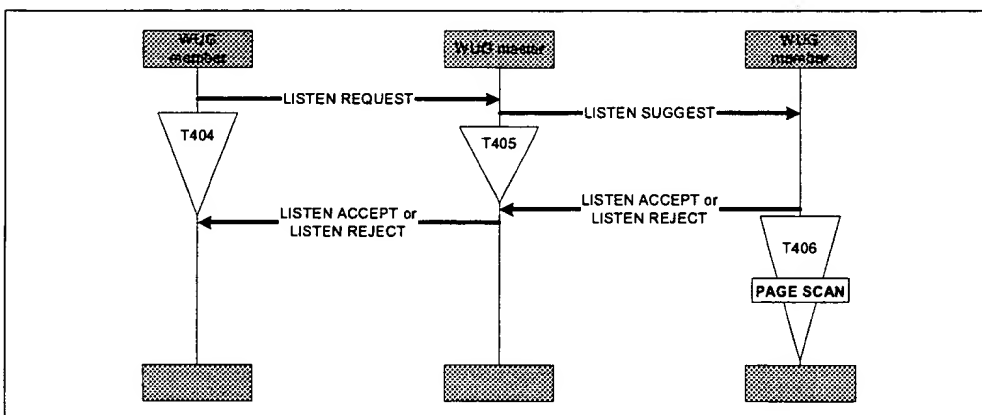


Figure 3.3: Fast inter-member access message flow

4 CONNECTIONLESS TCS (CL)

A connectionless TCS message can be used to exchange signalling information without establishing a TCS call. It is thus a connectionless service offered by TCS.

A connectionless TCS message is a CL INFO message (as defined in Section 6.3.1 on page 470).

A connection-oriented L2CAP channel between the Outgoing and Incoming Side shall be available before a CL INFO message can be sent.

Note: In the case of a connection-oriented channel, it may choose to delay the termination of the channel for a defined period to exchange more CL INFO messages.

Alternatively, in a multi-point configuration (see Section 1.2 on page 435), a connectionless L2CAP channel may be used and, if so, shall be available before a CL INFO can be sent.

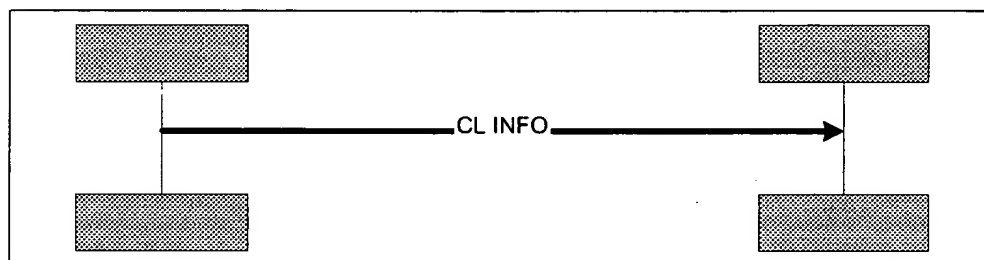


Figure 4.1: Connectionless TCS message flow

5 SUPPLEMENTARY SERVICES (SS)

The TCS provides explicit support for only one supplementary service, the Calling Line Identity (see Section 5.1 on page 456).

For supplementary services provided by an external network, using DTMF sequences for the activation/de-activation and interrogation of supplementary services, the DTMF start & stop procedure is supported (see Section 5.2 on page 456). This procedure allows both finite and infinite tone lengths.

Section 5.3 on page 458 specifies how a specific supplementary service, provided by an external network, called register recall is supported.

For other means of supplementary service control, no explicit support is specified. Support may be realized by either using the service call, or use the company specific information element, or a combination.

5.1 CALLING LINE IDENTITY

To inform the Incoming Side of the identity of the originator of the call, the Outgoing Side may include the calling party number information element (see Section 7.4.6 on page 481) in the SETUP message transferred as part of the call request.

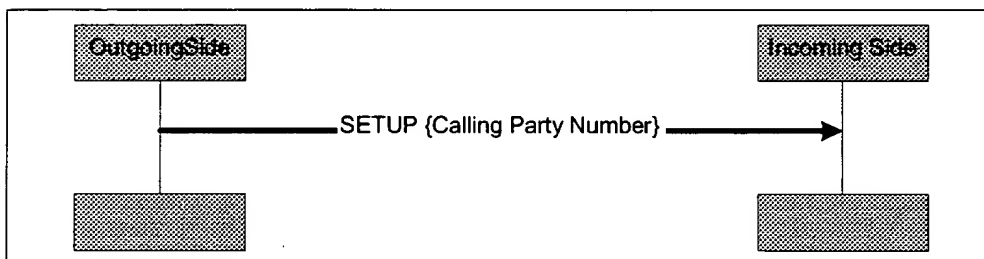


Figure 5.1: Calling line identity message flow

5.2 DTMF START & STOP

The DTMF start & stop procedure is supported to provide supplementary service control on PSTN type of networks.

In principle DTMF messages can be initiated by either (Outgoing or Incoming) Side; in practice, however, the Side (gateway) connected to the external PSTN network will be the recipient.

DTMF messages can be transmitted only in the active state of a call. Tone generation shall end when the call is disconnected.

5.2.1 Start DTMF request

A user may cause a DTMF tone to be generated; e.g. by depression of a key. The relevant action is interpreted as a requirement for a DTMF digit to be sent in a START DTMF message on an established signalling channel. This message contains the value of the digit to be transmitted (0, 1...9, A, B, C, D, *, #).

Only a single digit will be transferred in each START DTMF message.

5.2.2 Start DTMF response

The side receiving the START DTMF message will reconvert the received digit back into a DTMF tone which is applied toward the remote user, and return a START DTMF ACKNOWLEDGE message to the initiating side. This acknowledgment may be used to generate an indication as a feedback for a successful transmission.

If the receiving side cannot accept the START DTMF message, a START DTMF REJECT message will be sent to the initiating side, using cause #29, *Facility rejected*, indicating that sending DTMF is not supported by the external network.

5.2.3 Stop DTMF request

When the user indicates the DTMF sending should cease (e.g. by releasing the key) the initiating side will send a STOP DTMF message to the other side.

5.2.4 Stop DTMF response

Upon receiving the STOP DTMF message, the receiving side will stop sending the DTMF tone (if still being sent) and return a STOP DTMF ACKNOWLEDGE message to the initiating side.

5.2.5 Message flow

The figure below provides a view of the messages exchanged when a single DTMF tone needs to be generated.

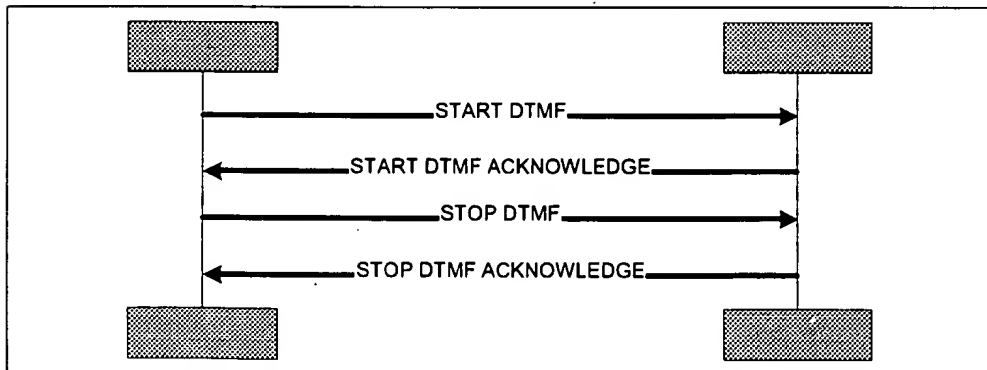


Figure 5.2: DTMF start & stop message flow

5.3 REGISTER RECALL

Register recall means to seize a register (with dial tone) to permit input of further digits or other action. In some markets, this is referred to as 'hook flash'. Register recall is supported by sending an INFORMATION message with a keypad facility information element, indicating 'register recall' (value 16H). Further digits are sent using the procedures as indicated in Section 5.2 above.

6 MESSAGE FORMATS

This section provides an overview of the structure of messages used in this specification, and defines the function and information contents (i.e. semantics) of each message.

Whenever a message is sent according to the procedures of Sections 2, 3 and 4, it shall contain the mandatory information elements, and optionally any combination of the optional information elements specified in this section for that message.

A message shall always be delivered in a single L2CAP packet. The start of a message (the LSB of the first octet) shall be aligned with the start of the L2CAP payload.

Each definition includes:

- a) A brief description of the message direction and use
- b) A table listing the information elements in order of their appearance in the message (same relative order for all message types)
- c) Indications for each information element in the table, specifying –
 - the section of this specification describing the information element
 - whether inclusion is mandatory ('M') or optional ('O')
 - the length (or length range) of the information element, where '*' denotes an undefined maximum length which may be application dependent.
- d) Further explanatory notes, as necessary

All message formats are denoted in octets.

6.1 CALL CONTROL MESSAGE FORMATS

6.1.1 ALERTING

This message is sent by the incoming side to indicate that the called user alerting has been initiated.

Message Type: ALERTING

Direction: incoming to outgoing

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Bearer capability	7.4.3	O Note 1)	4(26)
Progress indicator	7.4.13	O	2
SCO Handle	7.4.14	O	2
Destination CID	7.4.11	O	4
Company specific	7.4.9	O	3-*

Table 6.1: ALERTING message content

Note 1: Allowed only in the first message sent by the incoming side.

6.1.2 CALL PROCEEDING

This message is sent by the incoming side to indicate that the requested call establishment has been initiated and no more call establishment information will be accepted.

Message Type: CALL PROCEEDING

Direction: incoming to outgoing

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Bearer capability	7.4.3	O Note 1)	4(26)
Progress indicator	7.4.13	O	2
SCO Handle	7.4.14	O	2
Destination CID	7.4.11	O	4
Company specific	7.4.9	O	3-*

Table 6.2: CALL PROCEEDING message content

Note 1: Allowed only in the first message sent by the incoming sid .

6.1.3 CONNECT

This message is sent by the incoming side to indicate call acceptance by the called user.

Message Type: CONNECT

Direction: incoming to outgoing

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Bearer capability	7.4.3	O ^{Note 1)}	4(26)
SCO Handle	7.4.14	O	2
Company specific	7.4.9	O	3-*

Table 6.3: CONNECT message content

Note 1: Allowed only in the first message sent by the incoming side.

6.1.4 CONNECT ACKNOWLEDGE

This message is sent by the outgoing side to acknowledge the receipt of a CONNECT message.

Message Type: CONNECT ACKNOWLEDGE

Direction: outgoing to incoming

Information Element	Ref.	Type	Length
Message type	7.3	M	1
SCO Handle	7.4.14	O	2
Destination CID	7.4.11	O	4
Company specific	7.4.9	O	3-*

Table 6.4: CONNECT ACKNOWLEDGE message content

6.1.5 DISCONNECT

This message is sent by either side as an invitation to terminate the call.

Message Type: DISCONNECT

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Cause	7.4.7	O	2
Progress indicator	7.4.13	O	2
SCO Handle	7.4.14	O	2
Destination CID	7.4.11	O	4
Company specific	7.4.9	O	3-*

Table 6.5: DISCONNECT message content

6.1.6 INFORMATION

This message is sent by either side to provide additional information during call establishment (in case of overlap sending).

Message Type: INFORMATION

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Sending complete	7.4.15	O	1
Keypad facility	7.4.12	O	2
Called party number	7.4.5	O	3-*
Audio control	7.4.2	O	3-*
Company specific	7.4.9	O	3-*

Table 6.6: INFORMATION message content

6.1.7 PROGRESS

This message is sent by the incoming side to indicate the progress of a call in the event of interworking or by either side in the call with the provision of optional in-band information/patterns.

Message Type: PROGRESS

Direction: incoming to outgoing

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Progress indicator	7.4.13	M	2
SCO Handle	7.4.14	O	2
Destination CID	7.4.11	O	4
Company specific	7.4.9	O	3-*

Table 6.7: PROGRESS message content

6.1.8 RELEASE

This message is used to indicate that the device sending the message had disconnected the channel (if any) and intends to release the channel, and that receiving device should release the channel after sending RELEASE COMPLETE.

Message Type: RELEASE

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Cause	7.4.7	O ^{Note 1)}	2
Company specific	7.4.9	O	3-*

Table 6.8: RELEASE message content

Note 1: Mandatory in the first call clearing message.

6.1.9 RELEASE COMPLETE

This message is used to indicate that the device sending the message has released the channel (if any), and that the channel is available for re-use.

Message Type: RELEASE COMPLETE

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Cause	7.4.7	O ^{Note 1)}	2
Company specific	7.4.9	O	3-*

Table 6.9: RELEASE COMPLETE message content

Note 1: Mandatory in the first call clearing message.

6.1.10 SETUP

This message is sent by the outgoing side to initiate call establishment.

Message Type:

Direction:

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Call class	7.4.4	M	2
Sending complete	7.4.15	O	1
Bearer capability	7.4.3	O	4(26)
Signal	7.4.16	O	2
Calling party number	7.4.6	O	3-*
Called party number	7.4.5	O	3-*
Company specific	7.4.9	O	3-*

Table 6.10: SETUP message content

6.1.11 SETUP ACKNOWLEDGE

This message is sent by the incoming side to indicate that call establishment has been initiated, but additional information may be required.

Message Type: SETUP ACKNOWLEDGE

Direction: incoming to outgoing

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Bearer capability	7.4.3	O ^{Note 1)}	4(26)
Progress indicator	7.4.13	O	2
SCO Handle	7.4.14	O	2
Destination CID	7.4.11	O	4
Company specific	7.4.9	O	3-*

Table 6.11: SETUP ACKNOWLEDGE message content

Note 1: Allowed only in the first message sent by the incoming side.

6.1.12 Start DTMF

This message contains the digit the other side should reconvert back into a DTMF tone, which is then applied towards the remote user.

Message Type: Start DTMF

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Keypad facility	7.4.12	M	2

Table 6.12: Start DTMF message content

6.1.13 Start DTMF Acknowledge

This message is sent to indicate the successful initiation of the action required by the Start DTMF message.

Message Type: Start DTMF Acknowledge

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Keypad facility	7.4.12	M	2

Table 6.13: Start DTMF Acknowledge message content

6.1.14 Start DTMF Reject

This message is sent to indicate that the other side cannot accept the Start DTMF message.

Message Type: Start DTMF Reject

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Cause	7.4.7	O	2

Table 6.14: Start DTMF Reject message content

6.1.15 Stop DTMF

This message is used to stop the DTMF tone sent towards the remote user.

Message Type: Stop DTMF

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1

Table 6.15: Stop DTMF message content

6.1.16 Stop DTMF Acknowledge

This message is sent to indicate that the sending of the DTMF tone has been stopped.

Message Type: Stop DTMF Acknowledge

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Keypad facility	7.4.12	M	2

Table 6.16: Stop DTMF Acknowledge message content

6.2 GROUP MANAGEMENT MESSAGE FORMATS

6.2.1 ACCESS RIGHTS REQUEST

This message is sent by the initiating side to obtain access rights.

Message Type: ACCESS RIGHTS REQUEST

Direction:

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Company specific	7.4.9	O	3-*

Table 6.17: ACCESS RIGHTS REQUEST message content

6.2.2 ACCESS RIGHTS ACCEPT

This message is sent by the responding side to indicate granting of access rights.

Message Type: ACCESS RIGHTS ACCEPT

Direction:

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Company specific	7.4.9	O	3-*

Table 6.18: ACCESS RIGHTS ACCEPT message content

6.2.3 ACCESS RIGHTS REJECT

This message is sent by the responding side to indicate denial of access rights.

Message Type: ACCESS RIGHTS REJECT

Direction:

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Company specific	7.4.9	O	3-*

Table 6.19: ACCESS RIGHTS REJECT message content

6.2.4 INFO SUGGEST

This message is sent by the WUG master to indicate that a change has occurred in the WUG configuration.

Message Type: INFO SUGGEST

Direction: WUG master to WUG member

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Configuration Data	7.4.10	M	*
Company specific	7.4.9	O	3-*

Table 6.20: INFO SUGGEST message content

6.2.5 INFO ACCEPT

This message is sent by the WUG member to indicate the acceptance of the updated WUG configuration.

Message Type: INFO ACCEPT

Direction: WUG member to WUG master

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Company specific	7.4.9	O	3-*

Table 6.21: INFO ACCEPT message content

6.2.6 LISTEN REQUEST

This message is sent by a WUG member to indicate to the WUG master the request for a Fast inter-member access to the indicated WUG member.

Message Type: LISTEN REQUEST

Direction: WUG member to WUG master

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Called party number	7.4.6	M	3-*
Company specific	7.4.9	O	3-*

Table 6.22: LISTEN REQUEST message content

6.2.7 LISTEN SUGGEST

This message is sent by a WUG master to indicate to the WUG member the request for a Fast inter-member access.

Message Type: LISTEN SUGGEST

Direction: WUG master to WUG member

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Company specific	7.4.9	O	3-*

Table 6.23: LISTEN SUGGEST message content

6.2.8 LISTEN ACCEPT

This message is sent to indicate the acceptance of the previous request for a Fast inter-member access.

Message Type: LISTEN ACCEPT

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Clock offset	7.4.8	O	4
Company specific	7.4.9	O	3-*

Table 6.24: LISTEN ACCEPT message content

6.2.9 LISTEN REJECT

This message is sent to indicate the rejection of the previous request for a Fast inter-member access.

Message Type: LISTEN REJECT

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Cause	7.4.7	O	2
Company specific	7.4.9	O	3-*

Table 6.25: LISTEN REJECT message content

6.3 TCS CONNECTIONLESS MESSAGE FORMATS

6.3.1 CL INFO

This message is sent by either side to provide additional information in a connectionless manner.

Message Type: CL INFO

Direction: both

Information Element	Ref.	Type	Length
Message type	7.3	M	1
Audio control	7.4.2	O	3-*
Company specific	7.4.9	O	3-*

Table 6.26: CL INFO message content

7 MESSAGE CODING

The figures and text in this section describe message contents. Within each octet, the bit designated 'bit 1' is transmitted first, followed by bit 2, 3, 4, etc. Similarly, the octet shown at the top of the figure is sent first.

Whenever a message is sent, according to the procedures of Sections 2, 3 and 4, it shall be coded as specified in this section.

7.1 OVERVIEW

The coding rules follow ITU-T Recommendation Q.931, but is tailored to the specific needs of TCS.

Every message consists of:

- a) Protocol discriminator
- b) Message type, and
- c) Other information elements, as required

The Protocol discriminator and Message type is part of every TCS message, while the other information elements are specific to each message type.

8	7	6	5	4	3	2	1	
Protocol discriminator				Message type				octet 1
Other information elements as required								octet 2

Table 7.1: General message format

A particular information element shall be present only once in a given message.

The term 'default' implies that the value defined shall be used in the absence of any assignment or negotiation of alternative values.

For notation purposes – when a field extends over more than one octet, the order of bit values progressively decreases as the octet number increases. The least significant bit of the field is represented by the lowest numbered bit of the highest-numbered octet of the field. In general, bit 1 of each octet contains the least significant bit of a field.

7.2 PROTOCOL DISCRIMINATOR

The purpose of the protocol discriminator is to distinguish the TCS messages into different functional groups. The protocol discriminator is the first part of every message.

The protocol discriminator is coded according to Figure 7.1 and Table 7.2.

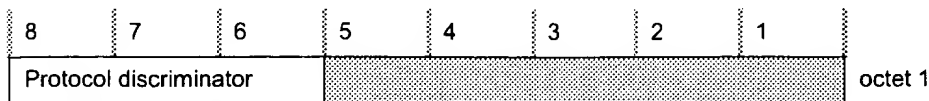


Figure 7.1: Protocol discriminator

Bits			
8	7	6	
0	0	0	Bluetooth TCS Call Control
0	0	1	Bluetooth TCS Group management
0	1	0	Bluetooth TCS Connectionless
			All other values reserved

Table 7.2: Protocol discriminator

7.3 MESSAGE TYPE

The purpose of the message type is to identify the function of the message being sent.

The Message type is the first part of every message and it is coded as shown in Figure 7.2 and Table 7.3.

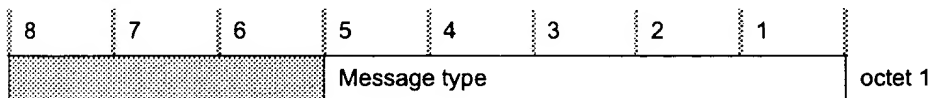


Figure 7.2: Message type

Bits					
5	4	3	2	1	
					Call Control messages
					Call Establishment
0	0	0	0	0	ALERTING

Table 7.3: Message type

Bits					
5	4	3	2	1	
0	0	0	0	1	CALL PROCEEDING
0	0	0	1	0	CONNECT
0	0	0	1	1	CONNECT ACKNOWLEDGE
0	0	1	0	0	PROGRESS
0	0	1	0	1	SETUP
0	0	1	1	0	SETUP ACKNOWLEDGE
					<i>Call clearing</i>
0	0	1	1	1	DISCONNECT
0	1	0	0	0	RELEASE
0	1	0	0	1	RELEASE COMPLETE
					<i>Miscellaneous</i>
0	1	0	1	0	INFORMATION
1	0	0	0	0	START DTMF
1	0	0	0	1	START DTMF ACKNOWLEDGE
1	0	0	1	0	START DTMF REJECT
1	0	0	1	1	STOP DTMF
1	0	1	0	0	STOP DTMF ACKNOWLEDGE
					<i>Group management messages</i>
0	0	0	0	0	INFO SUGGEST
0	0	0	0	1	INFO ACCEPT
0	0	0	1	0	LISTEN REQUEST
0	0	0	1	1	LISTEN ACCEPT
0	0	1	0	0	LISTEN SUGGEST
0	0	1	0	1	LISTEN REJECT
0	0	1	1	0	ACCESS RIGHTS REQUEST
0	0	1	1	1	ACCESS RIGHTS ACCEPT
0	1	0	0	0	ACCESS RIGHTS REJECT
					<i>Connectionless messages</i>
0	0	0	0	0	CL INFO

Table 7.3: Message type

7.4 OTHER INFORMATION ELEMENTS

7.4.1 Coding rules

The coding of other information elements follows the coding rules described below.

- Three categories of information elements are defined:
- a) single octet information elements (see Figure 7.3 on page 474)
 - b) double octet information element (see Figure 7.4 on page 474)
 - c) variable length information elements (see Figure 7.5 on page 474).

Table 7.4 on page 474 summarizes the coding of the information element identified bits for those information elements used in this specification.

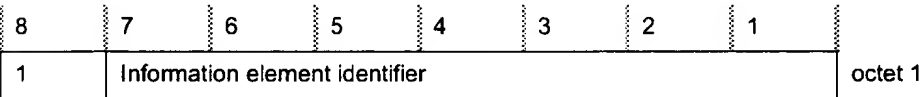


Figure 7.3: Single octet information element format

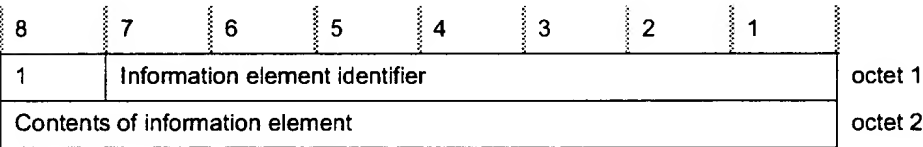


Figure 7.4: Double octet information element format

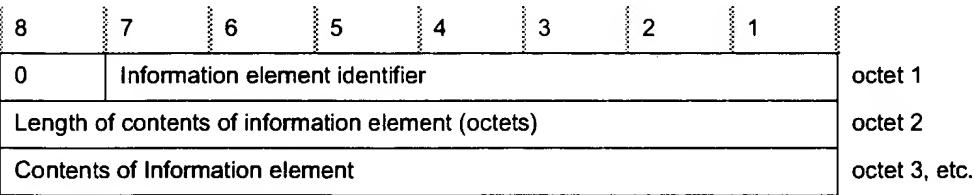


Figure 7.5: Variable length information element format

Coding								Ref.	Max Length (octets)
8	7	6	5	4	3	2	1		
1									
	0	1	0	0	0	0	1	7.4.15	1
1									

Table 7.4: Information element identifier coding

Coding								Ref.	Max Length (octets)	
8	7	6	5	4	3	2	1			
	1	0	0	0	0	0	0	Call class	7.4.4	2
	1	0	0	0	0	0	1	Cause	7.4.7	2
	1	0	0	0	0	1	0	Progress indicator	7.4.13	2
	1	0	0	0	0	1	1	Signal	7.4.16	2
	1	0	0	0	1	0	0	Keypad facility	7.4.12	2
	1	0	0	0	1	0	1	SCO handle	7.4.14	2
0								Variable length information elements		
	0	0	0	0	0	0	0	Clock offset	7.4.8	4
	0	0	0	0	0	0	1	Configuration data	7.4.2	*
	0	0	0	0	0	1	0	Bearer capability	7.4.3	4(26)
	0	0	0	0	0	1	1	Destination CID	7.4.11	4
	0	0	0	0	1	0	0	Calling party number	7.4.6	*
	0	0	0	0	1	0	1	Called party number	7.4.5	*
	0	0	0	0	1	1	0	Audio control	7.4.2	*
	0	0	0	0	1	1	1	Company specific	7.4.9	*

Table 7.4: Information element identifier coding

The descriptions of the information elements below are organized in alphabetical order. However, there is a particular order of appearance for each information element in a message. The code values of the information element identifier for the variable length formats are assigned in ascending numerical order, according to the actual order of appearance of each information element in a message. This allows the receiving devices to detect the presence or absence of a particular information element without scanning through an entire message.

Where the description of information elements in this specification contains spare bits, these bits are indicated as being set to '0'. In order to allow compatibility with future implementation, messages should not be rejected simply because a spare bit is set to '1'.

The second octet of a variable length information element indicates the total length of the contents of that information element regardless of the coding of the first octet (i.e. the length is calculated starting from octet 3). It is the binary coding of the number of octets of the contents, with bit 1 as the least significant bit (2^0).

An optional variable-length information element may be present, but empty (zero length). The receiver should interpret this as if that information element

was absent. Similarly, an absent information element should be interpreted by the receiver as if that information element was empty.

7.4.2 Audio control

The purpose of the Audio control information elements is to indicate information relating to the control of audio.

8	7	6	5	4	3	2	1	Octets
0	0	0	0	0	1	1	0	1
Length of contents of information element (octets)								2
Control information								3

Figure 7.6:

Control information (octet 3)							
Bits							
7	6	5	4	3	2	1	
0	0	0	0	0	0	0	Volume increase
0	0	0	0	0	0	1	Volume decrease
0	0	0	0	0	1	0	Microphone gain increase
0	0	0	0	0	1	1	Microphone gain decrease
0	X	X	X	X	X	X	Reserved for Bluetooth standardization
1	X	X	X	X	X	X	Company specific

Table 7.5: Audio Control information element coding

7.4.3 Bearer capability

The purpose of the Bearer capability information elements is to indicate a requested or available bearer service.

If this information element is absent, the default Bearer capability is Link type Synchronous Connection-Oriented with packet type HV3, using CVSD coding for the User information layer 1.

8	7	6	5	4	3	2	1	Octets
0	0	0	0	0	0	1	0	1
Length of contents of information element (octets)								2
Link type								3

Figure 7.7:

Link type element coding = 00000000 (SCO)

User information layer 1	Packet type	4
--------------------------	-------------	---

Figure 7.8:

Link type element coding = 00000001 (ACL)

Flags		4
Service type		5
Token Rate		6
		7
		8
		9
Token Bucket Size (bytes)		10
		11
		12
		13
Peak Bandwidth (bytes/second)		14
		15
		16
		17
Latency (microseconds)		18
		19
		20
		21
Delay Variation (microseconds)		22
		23
		24
		25
User information layer 3	User information layer 2	26

Figure 7.9:

Note: the Quality of Service is repeated at TCS level, as only TCS has the knowledge of end-to-end Quality of Service requirements.

Link type (octet 3)									
Bits									
8	7	6	5	4	3	2	1		
0	0	0	0	0	0	0	0	Synchronous Connection-Oriented	
0	0	0	0	0	0	0	1	Asynchronous Connection-Less	
0	0	0	0	0	0	1	0	None	
All other values are reserved									
Octet 4 coding (Link type element coding = 000000000)									
Packet type (octet 4)									
Bits									
5	4	3	2	1					
0	0	1	0	1	HV1				
0	0	1	1	0	HV2				
0	0	1	1	1	HV3				
0	1	0	0	0	DV				
All other values are reserved									
User information layer 1 (octet 4)									
Bits									
8	7	6							
0	0	1	CVSD						
0	1	0	PCM A-law						
0	1	1	PCM μ -law						
All other values reserved									
Octets 4-26 coding (Link type element coding = 000000001)									
The details of the coding Octets 4-25 can be found in L2CAP, see L2CAP, Section 6 on page 289									
User information layer 2 (octet 26)									
Bits									
4	3	2	1						
0	0	0	0	RFCOMM over L2CAP					
All other values are reserved									
User information layer 3 (octet 26)									
Bits									
8	7	6	5						
0	0	0	0	Not specified					
0	0	0	1	PPP					
0	0	1	0	IP					
All other values reserved									
Octet 4 coding (Link type element coding = 000000010)									
Octet 4 is absent									

Table 7.6: Bearer capability information element coding

7.4.4 Call class

The purpose of the Call class is to indicate the basic aspects of the service requested. This element allows the user to indicate the use of default attributes, thereby reducing the length of the set-up message.

8	7	6	5	4	3	2	1	Octets
1	1	0	0	0	0	0	0	1
Call Class								2

Figure 7.10:

Call class (octet 2)								
Bits								
8	7	6	5	4	3	2	1	
0	0	0	0	0	0	0	0	External call
0	0	0	0	0	0	0	1	Intercom call
0	0	0	0	0	0	1	0	Service call
0	0	0	0	0	0	1	1	Emergency call
All other values reserved								

Table 7.7: Call class information element coding

Note

- An external call is a call to/from an external network; e.g. the PSTN.
- An intercom call is a call between Bluetooth devices.
- A service call is a call for configuration purposes.
- An emergency call is an external call using a dedicated emergency call number, using specific properties.

7.4.5 Called party number

The purpose of the Called party number information element is to identify the called party of a call.

8	7	6	5	4	3	2	1	Octets
0	0	0	0	0	1	0	1	1
Length of contents of information element (octets)								2
0	Type of number			Numbering plan identification				3
0	Number digits (IA5 characters) (Note)							4 etc.

Note – The number digits appear in multiple octet 4's in the same order in which they would be entered, that is, the number digit which would be entered first is located in the first octet 4.

Figure 7.11:

Type of number (octet 3)				
Bits				
7	6	5		
0	0	0	Unknown	
0	0	1	International number	
0	1	0	National number	
0	1	1	Network specific number	
1	0	0	Subscriber number	
1	1	0	Abbreviated number	
1	1	1	Reserved for extension	
All other values are reserved				
Numbering plan identification (octet 3)				
Bits				
4	3	2	1	
0	0	0	0	Unknown
0	0	0	1	ISDN/telephony numbering plan E.164
0	0	1	1	Data numbering plan Rec. X.121
0	1	0	0	Reserved
1	0	0	0	National standard numbering plan
1	0	0	1	Private numbering plan
All other values are reserved				

Table 7.8: Called party information element coding

7.4.6 Calling party number

The purpose of the Calling party number information element is to identify the origin of a call.

8	7	6	5	4	3	2	1	Octets
0	0	0	0	0	1	0	0	1
Length of contents of information element (octets)								2
0	Type of number			Numbering plan identification				3
0	Presentation indicator		0	0	0	Screening indicator		4
0	Number digits (IA5 characters)							5 etc.

Figure 7.12:

Type of number (octet 3)				
Bits				
7	6	5		
0	0	0	Unknown	
0	0	1	International number	
0	1	0	National number	
0	1	1	Network specific number	
1	0	0	Subscriber number	
1	1	0	Abbreviated number	
1	1	1	Reserved for extension	
All other values are reserved				
Numbering plan identification (octet 3)				
Bits				
4	3	2	1	
0	0	0	0	Unknown
0	0	0	1	ISDN/telephony numbering plan E.164
0	0	1	1	Data numbering plan Rec. X.121
0	1	0	0	Reserved
1	0	0	0	National standard numbering plan
1	0	0	1	Private numbering plan
All other values are reserved				
Presentation indicator (octet 4)				
Bits				
7	6			
0	0	Presentation allowed		
0	1	Presentation restricted		
1	0	Number not available due to interworking		
1	1	Reserved		
All other values are reserved				

Table 7.9: Calling party information element coding

Screening indicator (octet 4)		
Bits		
2	1	
0	0	User-provided, not screened
0	1	User-provided, verified and passed
1	0	User-provided, verified and failed
1	1	Network provided
All other values are reserved		

Table 7.9: Calling party information element coding

7.4.7 Cause

The purpose of the Cause is to indicate the remote side of the cause of the failure of the requested service.

8	7	6	5	4	3	2	1	Octets
1	1	0	0	0	0	0	1	1
Cause value								2

Figure 7.13:

Cause (octet 2)								
Bits								
8	7	6	5	4	3	2	1	
0	These 7 bits are coded alike the Cause value subfield defined in Section 2.2.5 of ITU-T Recommendation Q.850[2].							

Table 7.10: Cause information element coding

7.4.8 Clock offset

The purpose of the Clock offset information element is to indicate the Bluetooth clock offset used.

8	7	6	5	4	3	2	1	Octets
0	0	0	0	0	0	0	0	1
Length of contents of information element (octets)								2
Clock offset								3
								4

Figure 7.14:

Clock offset coding (octet 3 and 4)															
Bits (octet 3)								Bits (octet 4)							
8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Contains bits 16-2 of Bluetooth clock															

Table 7.11: Clock offset information element coding

7.4.9 Company specific

The purpose of the Company specific information element is to send non-standardized information.

8	7	6	5	4	3	2	1	Octets
0	0	0	0	0	1	1	1	1
Length of contents of information element (octets)								2
Company Identification								3
Company Identification								4
Company specific contents								L+2

Figure 7.15:

Company identification coding (octet 3 and octet 4)															
Bits (octet 3)								Bits (octet 4)							
8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
All other values are reserved															

Table 7.12: Company specific information element coding

7.4.10 Configuration data

The purpose of the Configuration data information element is to indicate the Configuration data.

8	7	6	5	4	3	2	1	Octets
0	0	0	0	0	0	0	1	1
Length of contents of information element (octets)								2
0	Internal number of WUG member 1 (IA5 characters)							3
0	Internal number of WUG member 1 (IA5 characters)							4
Bluetooth address of WUG member 1								5
								...
								10
Link key to be used towards WUG member 1								11
								..
								26
.....								
0	Internal number of WUG member n (IA5 character)							$3+((n-1)*24)$
0	Internal number of WUG member n (IA5 character)							$4+((n-1)*24)$
Bluetooth address of WUG member n								$5+((n-1)*24)$
								...
								$10+((n-1)*24)$
Link key to be used towards WUG member n								$11+((n-1)*24)$
								..
								$2+(n*24)$

Note – The internal number (2 digits) appears in octets 3 and 4 in the same order in which they would be entered; that is, the number digit which would be entered first is located in octet 3.

Note – The octets 3-26 are repeated for all n WUG members.

Figure 7.16:

7.4.11 Destination CID

The purpose of the Destination CID information element is to enable the remote side to associate the established L2CAP channel with the ongoing call. The Destination CID is identical to the Destination CID (DCID) exchanged in the Configuration Request packet (see L2CAP, Section 5.4 on page 280).

8	7	6	5	4	3	2	1	Octets
0	0	0	0	0	0	1	1	1
Length of contents of information element (octets)								2
DCID byte 1								3
DCID byte 0								4

Figure 7.17:

7.4.12 Keypad facility

The purpose of the Keypad facility information element is to convey IA5 characters; e.g. entered by means of a terminal keypad.

8	7	6	5	4	3	2	1	Octets
1	1	0	0	0	1	0	0	1
0	Keypad facility information (IA5 character)							2

Figure 7.18:

7.4.13 Progress indicator

The purpose of the Progress indicator information element is to describe an event that has occurred during the life of a call.

8	7	6	5	4	3	2	1	Octets
1	1	0	0	0	0	1	0	1
0	Progress description							2

Figure 7.19:

Progress information (octet 2)							
Bits							
7	6	5	4	3	2	1	
0	0	0	1	0	0	0	In-band information or appropriate pattern is now available
All other values reserved							

Table 7.13: Progress indicator information element coding

7.4.14 SCO Handle

The purpose of the SCO handle information element is to enable the remote side to associate the established SCO link with the ongoing call. The SCO handle is identical to the SCO handle exchanged in the LMP_SCO_link_req sent by the piconet master (see LMP, Section 3.21 on page 219).

8	7	6	5	4	3	2	1	Octets
1	1	0	0	0	1	0	1	1
SCO handle value								2

Figure 7.20:

7.4.15 Sending complete

The purpose of the Sending complete information element is to optionally indicate completion of called party number.

8	7	6	5	4	3	2	1	Octet
1	0	1	0	0	0	0	1	1

Figure 7.21:

7.4.16 Signal

The purpose of the Signal information element is to convey information to a user regarding tones and alerting signals.

8	7	6	5	4	3	2	1	Octets
1	1	0	0	0	0	1	1	1
Signal value								2

Figure 7.22:

Signal value (octet 2)								
	Bits							
8	7	6	5	4	3	2	1	
0	1	0	0	0	0	0	0	External call
0	1	0	0	0	0	0	1	Internal call
0	1	0	0	0	0	1	0	Call back
0	X	X	X	X	X	X	X	Reserved for Bluetooth standardization
1	X	X	X	X	X	X	X	Company specific

Table 7.14: Signal information element coding

8 MESSAGE ERROR HANDLING¹

8.1 PROTOCOL DISCRIMINATION ERROR

When a message is received with a protocol discriminator coded other than the ones defined in Section 7.2 on page 472, that message shall be ignored.

8.2 MESSAGE TOO SHORT OR UNRECOGNIZED

When a message is received that is too short to contain a complete message type information element, that message shall be ignored.

When a message is received that contains a complete message type information element, but with a value which is not recognized as a defined message type, that message shall be ignored.

8.3 MESSAGE TYPE OR MESSAGE SEQUENCE ERRORS

Whenever an unexpected message, except RELEASE or RELEASE COMPLETE message is received in any state other than the Null state, that message shall be ignored.

When an unexpected RELEASE message is received, the receiving side shall disconnect and release the bearer channel if established, return a RELEASE COMPLETE message, stop all timers, and enter the Null state.

When an unexpected RELEASE COMPLETE message is received, the receiving side shall disconnect and release the bearer channel if established, stop all timers, and enter the Null state.

8.4 INFORMATION ELEMENT ERRORS

The information elements in a message shall appear (if present for information elements indicated as optional) in the exact order as indicated in Section 6.

When a message is received which misses a mandatory information element, or which contains a mandatory information element with invalid content, the message shall be ignored.

In case the error occurred with a mandatory information element in a SETUP message, a RELEASE COMPLETE message shall be returned, either with cause #96, *mandatory information element is missing*, or with cause #100, *invalid information element contents*.

1. In this section, when it is stated to ignore a certain message or part of a message (information element), this shall be interpreted as to do nothing – as if the (part of the) message had never been received.

When a message is received which has an unrecognized information element, or has an optional information element with an invalid content, or has a recognized information element not defined to be contained in that message, the receiving side shall ignore the information element.

Information elements with a length exceeding the maximum length (as given in Section 7 on page 471) shall be treated as an information element with invalid content.

9 PROTOCOL PARAMETERS

9.1 PROTOCOL TIMERS

Timer name	Value
T301	Minimum 3 minutes
T302	15 seconds
T303	20 seconds
T304	30 seconds
T305	30 seconds
T308	4 seconds
T310	30 –120 seconds
T313	4 seconds
T401	8 second
T402	8 seconds
T403	4 second
T404	2.5 seconds
T405	2 seconds
T406	20 seconds

Table 9.1: Timer values

10 REFERENCES

- [1] Q.931, "Digital Subscriber Signalling System No. 1(DSS 1) – ISDN User-Network interface Layer 3 Specification for Basic Call Control", 03/93
- [2] Q.850, "Digital Subscriber Signalling System No. 1 General – Usage of cause of location in the Digital Subscriber Signalling system No. 1 and the signalling system No. 7 ISDN User Part", 03/93

11 LIST OF FIGURES

Figure 1.1:	TCS within the Bluetooth stack	435
Figure 1.2:	Point-to-point signalling in a single-point configuration	436
Figure 1.3:	Signalling in a multi-point configuration.....	436
Figure 1.4:	TCS Architecture	437
Figure 2.1:	Call establishment message flow	445
Figure 2.2:	Call clearing message flow	448
Figure 3.1:	Obtain access rights message flow.....	451
Figure 3.2:	Configuration distribution message flow	452
Figure 3.3:	Fast inter-member access message flow.....	454
Figure 4.1:	Connectionless TCS message flow	455
Figure 5.1:	Calling line identity message flow	456
Figure 5.2:	DTMF start & stop message flow.....	457
Figure 7.1:	Protocol discriminator.....	472
Figure 7.2:	Message type.....	472
Figure 7.3:	Single octet information element format.....	474
Figure 7.4:	Double octet information element format	474
Figure 7.5:	Variable length information element format	474
Figure 7.6:	476
Figure 7.7:	476
Figure 7.8:	477
Figure 7.9:	477
Figure 7.10:	479
Figure 7.11:	480
Figure 7.12:	481
Figure 7.13:	482
Figure 7.14:	482
Figure 7.15:	483
Figure 7.16:	484
Figure 7.17:	485
Figure 7.18:	485
Figure 7.19:	485
Figure 7.20:	486
Figure 7.21:	486
Figure 7.22:	486
Figure A:	Full TCS State Diagram.....	493
Figure B:	Lean TCS State Diagram.....	494

12 LIST OF TABLES

Table 6.1:	ALERTING message content.....	460
Table 6.2:	CALL PROCEEDING message content	460
Table 6.3:	CONNECT message content.....	461
Table 6.4:	CONNECT ACKNOWLEDGE message content	461
Table 6.5:	DISCONNECT message content.....	462
Table 6.6:	INFORMATION message content.....	462
Table 6.7:	PROGRESS message content	463
Table 6.8:	RELEASE message content.....	463
Table 6.9:	RELEASE COMPLETE message content.....	464
Table 6.10:	SETUP message content	464
Table 6.11:	SETUP ACKNOWLEDGE message content	465
Table 6.12:	Start DTMF message content.....	465
Table 6.13:	Start DTMF Acknowledge message content	466
Table 6.14:	Start DTMF Reject message content.....	466
Table 6.15:	Stop DTMF message content.....	466
Table 6.16:	Stop DTMF Acknowledge message content.....	467
Table 6.17:	ACCESS RIGHTS REQUEST message content.....	467
Table 6.18:	ACCESS RIGHTS ACCEPT message content.....	467
Table 6.19:	ACCESS RIGHTS REJECT message content	468
Table 6.20:	INFO SUGGEST message content	468
Table 6.21:	INFO ACCEPT message content	468
Table 6.22:	LISTEN REQUEST message content	469
Table 6.23:	LISTEN SUGGEST message content	469
Table 6.24:	LISTEN ACCEPT message content	469
Table 6.25:	LISTEN REJECT message content.....	470
Table 6.26:	CL INFO message content	470
Table 7.1:	General message format	471
Table 7.2:	Protocol discriminator	472
Table 7.3:	Message type	472
Table 7.4:	Information element identifier coding.....	474
Table 7.5:	Audio Control information element coding.....	476
Table 7.6:	Bearer capability information element coding.....	478
Table 7.7:	Call class information element coding	479
Table 7.8:	Called party information element coding	480
Table 7.9:	Calling party information element coding.....	481
Table 7.10:	Cause information element coding	482
Table 7.11:	Clock offset information element coding.....	483
Table 7.12:	Company specific information element coding	483
Table 7.13:	Progress indicator information element coding.....	485
Table 7.14:	Signal information element coding.....	486
Table 9.1:	Timer values	489

APPENDIX 1 - TCS CALL STATES

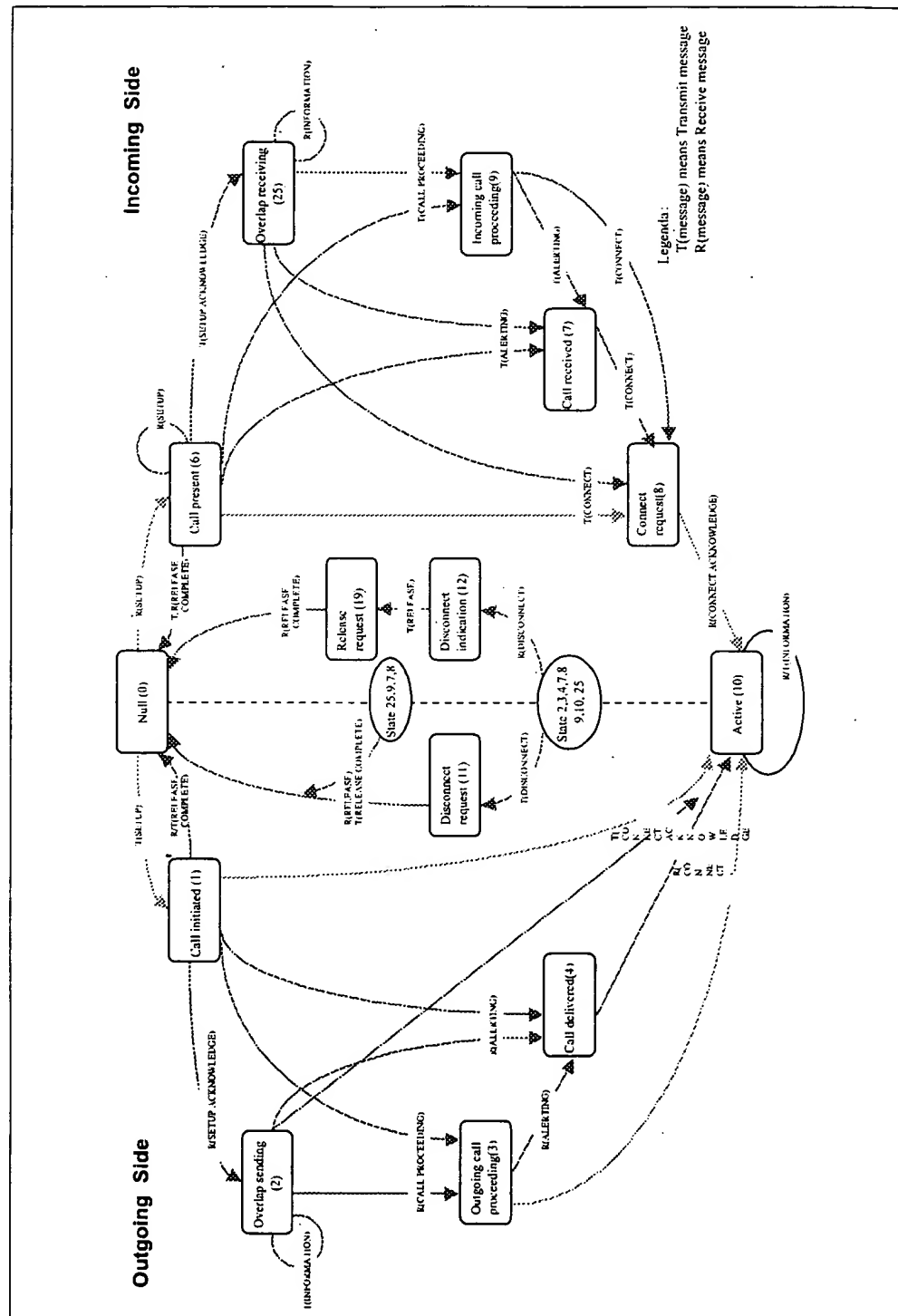


Figure A: Full TCS State Diagram

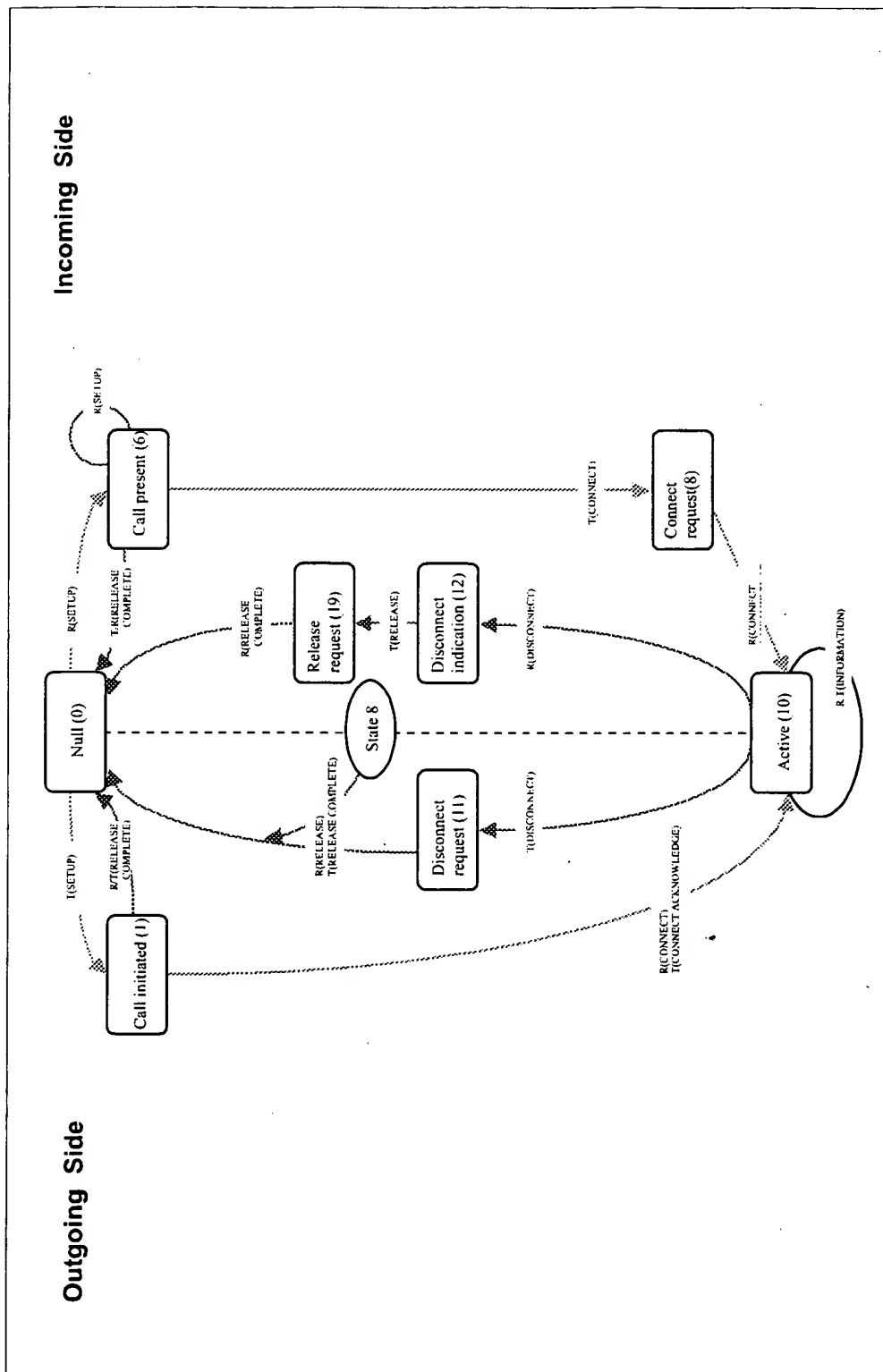


Figure B: Lean TCS State Diagram

Part F:4

**INTEROPERABILITY
REQUIREMENTS FOR BLUETOOTH
AS A WAP BEARER**

PPP Adaptation

Many of the characteristics of Bluetooth devices are shared with the target platforms for the Wireless Application Protocol. In some cases, the same device may be enabled for both types of communication. This document describes the interoperability requirements for using Bluetooth with PPP as the communications bearer for WAP protocols and applications.

CONTENTS

1	Introduction	499
1.1	Document Scope	499
2	The Use of WAP In the Bluetooth Environment	500
2.1	Value-added Services	500
2.2	Usage Cases	500
2.2.1	Briefcase Trick.....	500
2.2.2	Forbidden Message.....	501
2.2.3	WAP Smart Kiosk.....	501
3	WAP Services Overview	502
3.1	WAP Entities	502
3.1.1	WAP Client	502
3.1.2	WAP Proxy/Gateway	503
3.1.3	WAP Server.....	503
3.2	WAP Protocols	503
3.2.1	Wireless Datagram Protocol (WDP).....	504
3.2.2	Wireless Transaction Protocol (WTP)	504
3.2.3	Wireless Transport Layer Security (WTLS)	504
3.2.4	Wireless Session Protocol (WSP)	504
3.3	Contrasting WAP and Internet Protocols	504
3.3.1	UDP/WDP	504
3.3.2	WTP/TCP	505
3.3.3	WTLS/SSL.....	505
3.3.4	WSP/HTTP.....	505
3.3.5	WML/HTML	505
3.3.6	WMLScript/JavaScript.....	505

4	WAP in the Bluetooth Piconet	506
4.1	WAP Server Communications	506
4.1.1	Initiation by the Client Device.....	506
4.1.1.1	Discovery of Services	507
4.1.2	Termination by the Client Device.....	507
4.1.3	Initiation by the Server Device	507
4.1.3.1	Discovery of Services	508
4.2	Implementation of WAP for Bluetooth.....	508
4.2.1	WDP Management Entity.....	508
4.2.1.1	Asynchronous Notifications	508
4.2.1.2	Alternate Bearers.....	508
4.2.2	Addressing	509
4.3	Network Support for WAP.....	509
4.3.1	PPP/RFCOMM.....	509
5	Interoperability Requirements	511
5.1	Stage 1 – Basic Interoperability	511
5.2	Stage 2 – Advanced Interoperability.....	511
6	Service Discovery	512
6.1	SDP Service Records	512
6.2	SDP Protocol Data Units	514
6.3	Service discovery procedure	514
7	References.....	515

1 INTRODUCTION

1.1 DOCUMENT SCOPE

This document is intended for Bluetooth implementers who wish to take advantage of the dynamic, ad-hoc characteristics of the Bluetooth environment in providing access to value-added services using the WAP environment and protocols.

Bluetooth provides the physical medium and link control for communications between WAP client and server. This document describes how PPP may be used to achieve this communication.

The information contained in this document is not sufficient to allow the implementation of a general-purpose WAP client or server device. Instead, this document provides the following information:

- An overview of the use of WAP in the Bluetooth environment will explain how the concept of value-added services fits within the Bluetooth vision. Examples are given of how the WAP value-added services model can be used to fulfil specific Bluetooth usage models.
- The WAP Services Overview attempts to place the WAP environment in a familiar context. Each component of WAP is introduced, and is contrasted with equivalent Internet protocols (where applicable).
- A discussion of WAP in the Bluetooth Piconet describes how the particular structure of Bluetooth communications relates to WAP behaviors.
- Finally, the Interoperability Requirements describe the specific Bluetooth features that must be implemented in order to ensure interoperability between any two WAP enabled Bluetooth devices.

2 THE USE OF WAP IN THE BLUETOOTH ENVIRONMENT

2.1 VALUE-ADDED SERVICES

The presence of communications capabilities in a device is unlikely to be an end in itself. The end users are generally not as interested in the technology as in what the technology allows them to do.

Traditional telecommunications relies on voice communications as the single application of the technology, and this approach has been successful in the marketplace. As data communications services have become more widely available, there is increasing pressure to provide services that take advantage of those data capabilities.

The Wireless Application Protocol Forum was formed to create a standards-based framework, in which value-added data services can be deployed, ensuring some degree of interoperability.

2.2 USAGE CASES

The unique quality of Bluetooth, for the purposes of delivering value-added services, is the limited range of the communications link. Devices that incorporate Bluetooth are ideally suited for the receipt of location-dependent services. The following are examples of how the WAP client / server model can be applied to Bluetooth usage cases.

2.2.1 Briefcase Trick

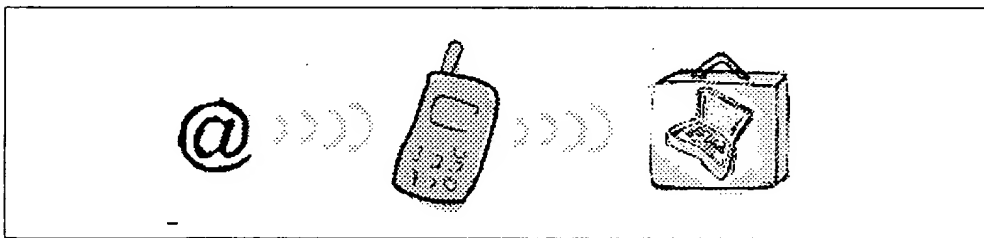


Figure 2.1: The 'Briefcase Trick' Hidden Computing Scenario

The Briefcase Trick usage case allows the user's laptop and mobile phone to communicate, without user intervention, in order to update the user's e-mail. The user can review the received messages from the handset, all without removing the laptop from its storage in a briefcase.

2.2.2 Forbidden Message

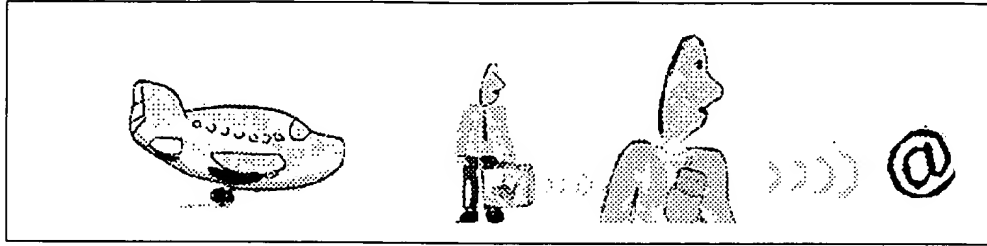


Figure 2.2: The 'Forbidden Message' Hidden Computing Scenario

The Forbidden Message usage case is similar to the briefcase trick. The user can compose messages in an environment where no dial-up connection is possible. At a later time the laptop wakes up, and checks the mobile phone to see if it is possible to send the pending messages. If the communications link is present, then the mail is transmitted.

2.2.3 WAP Smart Kiosk

The WAP Smart Kiosk usage case allows a user to connect a mobile PC or handheld device to communicate with a kiosk in a public location. The kiosk can provide information to the device that is specific to the user's location. For example, information on flights and gates in an airport, store locations in a shopping centre, or train schedules or destination information on a railway platform.

3 WAP SERVICES OVERVIEW

The Wireless Application Protocol is designed to provide Internet and Internet-like access to devices that are constrained in one or more ways. Limited communications bandwidth, memory, processing power, display capabilities and input devices are all factors driving the development of WAP. Although some devices may only exhibit some of the above constraints, WAP can still provide substantial benefit for those devices as well.

The WAP environment typically consists of three types of device: the WAP Client device, the WAP Proxy/gateway and WAP Server. In some cases the WAP Proxy/gateway may also include the server functionality.

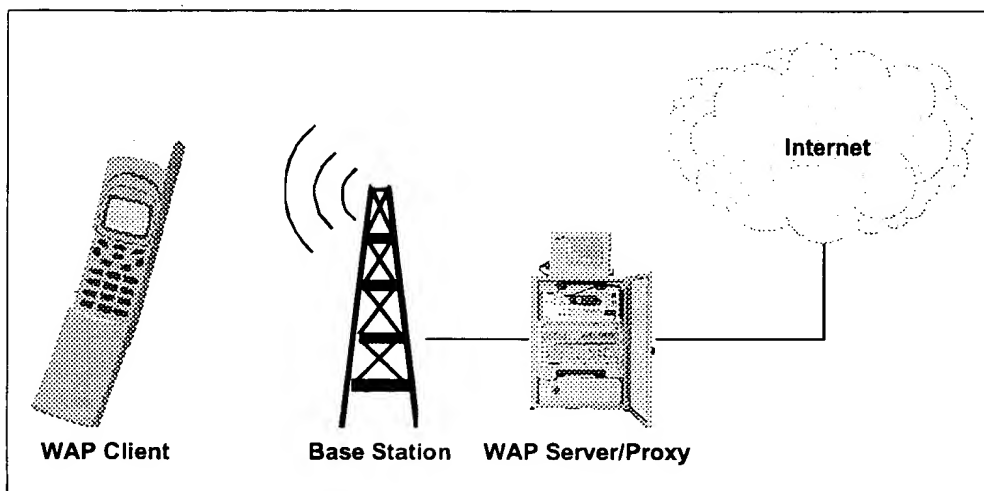


Figure 3.1: Typical WAP Environment

3.1 WAP ENTITIES

3.1.1 WAP Client

The WAP Client device is usually found in the hands of the end user. This device can be as powerful as a portable computer, or as compact as a mobile phone. The essential feature of the client is the presence of some type of display and some type of input device.

The WAP Client is typically connected to a WAP Proxy/gateway through a wireless network. (Figure 3.2 on page 503) This network may be based on any available technology. The WAP protocols allow the network to exhibit low reliability and high latency without interruption in service.

3.1.2 WAP Proxy/Gateway

The WAP Proxy/gateway acts as an interface between the wireless network, and the larger Internet. The primary functions of the proxy are to provide DNS name resolution services to WAP client devices and translation of Internet protocols and content formats to their WAP equivalents.

3.1.3 WAP Server

The WAP Server performs a function that is similar to a server in the Internet world. In fact, the WAP server is often an HTTP server. The server exists as a storage location for information that the user can access. This 'content' may include text, graphics, and even scripts that allow the client device to perform processing on behalf of the server.

The WAP Server logic may exist on the same physical device as the Proxy/gateway, or it may reside anywhere in the network that is reachable from the Proxy/gateway.

The server may fill the role of an HTTP server, a WSP server, or both.

3.2 WAP PROTOCOLS

The WAP environment consists of a layered protocol stack that is used to isolate the user agents from the details of the communications network. Figure 4.1 on page 506 illustrates the general architecture of the WAP protocol stack. Bluetooth will provide an additional data bearer service, appearing at the bottom of this diagram.

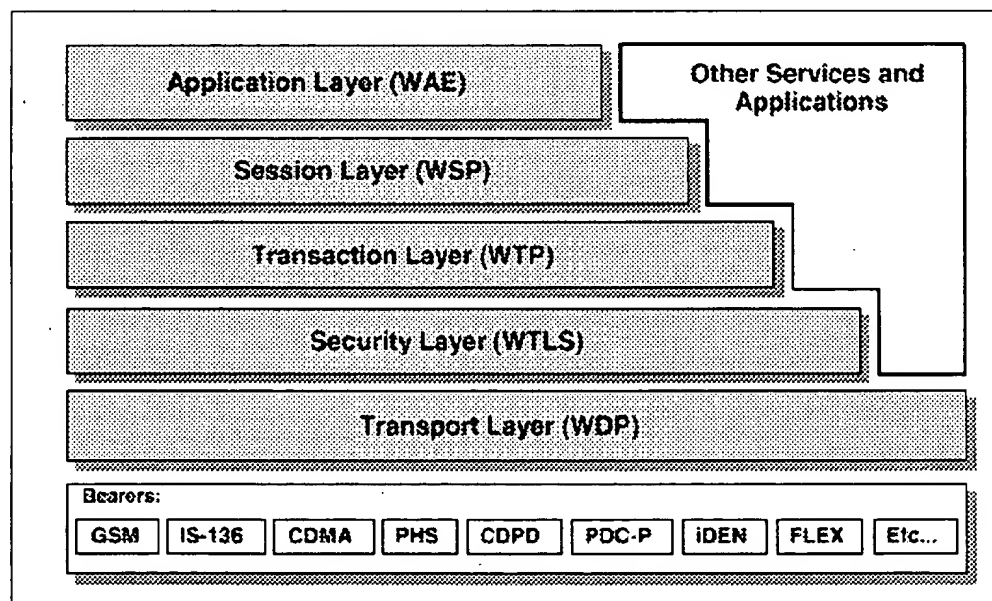


Figure 3.2: WAP Protocol Stack

3.2.1 Wireless Datagram Protocol (WDP)

The WDP layer provides a service interface that behaves as a socket-based UDP implementation. For a bearer service based on IP, then this layer is UDP. For bearer which do not provide a UDP service interface, then an implementation of WDP must be provided to act as an adaptation layer to allow socket-based UDP datagrams over the native bearer.

3.2.2 Wireless Transaction Protocol (WTP)

The WTP layer provides a reliable datagram service on top of the WDP (UDP) layer below.

3.2.3 Wireless Transport Layer Security (WTLS)

The WTLS layer is an optional component of the protocol stack that provides a secure data pipe between a client WSP session and its peer server WSP session. In the current version of the WAP specification, this session will terminate at the WAP server. There is currently a proposal before the WAP Forum for a proxy protocol, which will allow the intermediate WAP proxy to pass WTLS traffic across the proxy/gateway without decrypting the data stream.

3.2.4 Wireless Session Protocol (WSP)

The WSP layer establishes a relationship between the client application, and the WAP server. This session is relatively long-lived and able to survive service interruptions. The WSP uses the services of the WTP for reliable transport to the destination proxy/gateway.

3.3 CONTRASTING WAP AND INTERNET PROTOCOLS

The intent and implementation of the WAP protocol stack has many parallels with those of the Internet Engineering Task Force (IETF). The primary objective of the WAP Forum has been to make Internet content available to devices that are constrained in ways that make Internet protocols unsuitable for deployment.

This section compares the roles of the WAP protocol stack's layers with those of the IETF.

3.3.1 UDP/WDP

At the most basic layer, WAP and Internet protocols are the same. The WAP stack uses the model of a socket-based datagram (UDP) service as its transport interface.

Some Internet protocols also use the UDP service, but most actually use a connection-oriented stream protocol (TCP).

3.3.2 WTP/TCP

The wireless transport protocol (WTP) provides services that, in some respects, fill the same requirements as TCP. The Internet Transmission Control Protocol (TCP) provides a reliable, connection-oriented, character-stream protocol that is based on IP services. In contrast, WTP provides both reliable and unreliable, one-way and reliable two-way message transports. The transport is optimized for WAP's 'short request, long response' dialogue characteristic. WTP also provides message concatenation to reduce the number of messages transferred.

3.3.3 WTLS/SSL

The Wireless Transport Layer Security (WTLS) is derived from the Secure Sockets Layer (SSL) specification. As such, it performs the same authentication and encryption services as SSL.

3.3.4 WSP/HTTP

Session services in WAP are provided by the Wireless Session Protocol (WSP). This protocol incorporates the semantics and functionality of HTTP 1.1, while adding support for long-lived sessions, data push, suspend and resume. Additionally, the protocol uses compact encoding methods to adapt to narrow-band communications channels.

3.3.5 WML/HTML

The markup language used by WAP is a compact implementation that is similar to HTML, but optimized for use in hand-held devices. WML is an XML-defined markup language.

3.3.6 WMLScript/JavaScript

WAP also incorporates a scripting language that is similar to JavaScript, but adapted to the types of constrained devices that WAP is targeted for.

4 WAP IN THE BLUETOOTH PICONET

In many ways, Bluetooth can be used like other wireless networks with regard to WAP. Bluetooth can be used to provide a bearer for transporting data between the WAP Client and its adjacent WAP Server.

Additionally, Bluetooth's *ad hoc* nature provides capabilities that are exploited uniquely by the WAP protocols.

4.1 WAP SERVER COMMUNICATIONS

The traditional form of WAP communications involves a client device that communicates with a Server/Proxy device using the WAP protocols. In this case the Bluetooth medium is expected to provide a bearer service as specified by the WAP architecture.

4.1.1 Initiation by the Client Device

When a WAP client is actively 'listening' for available Bluetooth devices, it can discover the presence of a WAP server using Bluetooth's Service Discovery Protocol.

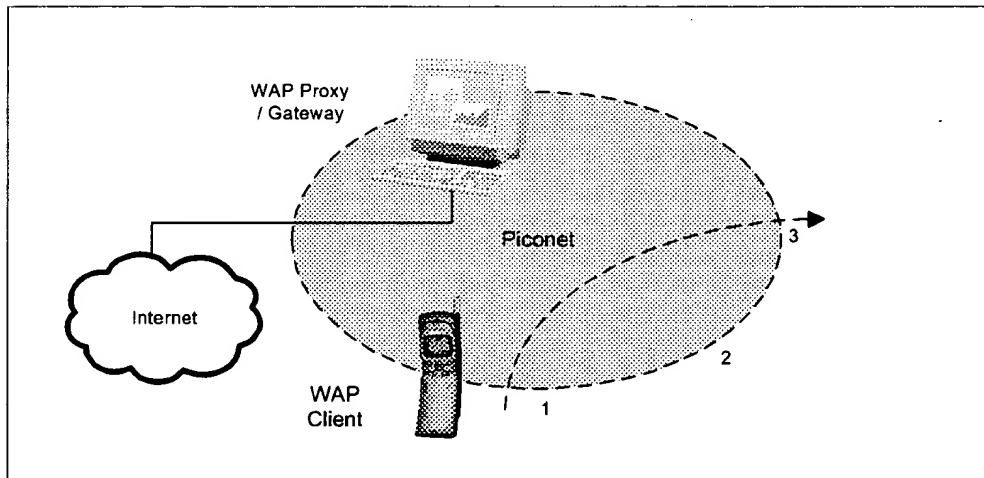


Figure 4.1: WAP Server / Proxy in Piconet

In Figure 4.1, stage 1 the WAP Client device is moving into range of the WAP Proxy/gateway's piconet. When the client detects the presence of the WAP proxy/gateway, it can automatically, or at the client's request, connect to the server.

4.1.1.1 Discovery of Services

The client must be able to determine the specific nature of the WAP proxy/gateway that it has detected. It is expected that the Bluetooth Service Discovery Protocol will be used to learn the following information about the server:

- Server Name – this is a user readable descriptive name for the server.
- Server Home Page Document Name – this is the home page URL for the server. This is optional.
- Server/Proxy Capability – indicates if the device is a WAP content server, a Proxy or both. If the device is a Proxy, it must be able to resolve URLs that are not local to the Server/Proxy device.

In Figure 4.1, stage 2, the device is communicating with the WAP proxy/gateway. All WAP data services normally available are possible.

4.1.2 Termination by the Client Device

In Figure 4.1, stage 3, the device is exiting the piconet. When the device detects that communication has been lost with the WAP proxy/gateway, it may optionally decide to resume communications using the information obtained at discovery.

For example, a client device that supports alternate bearers may query the alternate address information of the server when that capability is indicated. The information should be cached for later access because the client device may leave the piconet at any time, and that information will no longer be available.

In the WAP Smart Kiosk example above, if the user wishes to continue receiving information while out of Bluetooth range, the Kiosk would provide an Internet address to the client device. When Bluetooth communications are not possible, the device could use cellular packet data to resume the client-server session.

This capability is implementation-dependent, and is provided here for illustrative purposes only.

4.1.3 Initiation by the Server Device

An alternative method of initiating communications between a client and server is for the server to periodically check for available client devices. When the server device discovers a client that indicates that it has WAP Client capability, the server may optionally connect and push data to the client.

The client device has the option of ignoring pushed data at the end user's discretion.

4.1.3.1 Discovery of Services

Through the Bluetooth Service Discovery Protocol, the server can determine the following information about the client:

- Client Name – this is a friendly format name that describes the client device
- Client capabilities – this information allows the server to determine basic information regarding the client's Bluetooth-specific capabilities

4.2 IMPLEMENTATION OF WAP FOR BLUETOOTH

In order to effectively implement support for WAP over Bluetooth, certain capabilities must be considered.

4.2.1 WDP Management Entity

Associated with an instance of the WDP layer in the WAP Protocol Stack is an entity that is responsible for managing the services provided by that layer. The WDP Management Entity (WDP-ME) acts as an out-of-band mechanism for controlling the protocol stack.

4.2.1.1 Asynchronous Notifications

The WDP-ME will need to be able to generate asynchronous notifications to the application layer when certain events occur. Example notifications are:

- New Client Node Detected
- New Server Node Detected
- Client Node Signal Lost
- Server Node Signal Lost
- Server Push Detected (detected as unsolicited content)

Platform support for these events is implementation-specific. All of the listed events may be derived through the Bluetooth Host Controller Interface (page 517), with the exception of Server Push.

4.2.1.2 Alternate Bearers

An implementation of WAP on a particular device may choose to support multiple bearers. Methods of performing bearer selection are beyond the scope of this document. The procedure to be followed is implementation-dependent. See Section 4.1.2 above.

4.2.2 Addressing

Two basic types of addressing are being used in the WAP environment: User Addressing and Proxy/gateway Addressing. User addressing describes the location of objects within the network, and is independent of the underlying bearer. Proxy/Gateway Addressing describes the location of the WAP proxy/gateway that the device is communicating with. Proxy/Gateway addressing is dependent on the bearer type.

The end user deals mainly with Uniform Resource Locators (URL). These addresses are text strings that describe the document that is being accessed. Typically, the Proxy/gateway in conjunction with Internet Domain Name.

Servers resolve these strings into network addresses.

The address of the WAP Proxy/gateway is usually a static value that is configured by the user or network operator. When the user enters a URL, the request is forwarded to the configured WAP proxy/gateway. If the URL is within the domain of a co-located server, then it indicates that the document is actually WAP content. If the URL is outside of the WAP proxy/gateway's domain, then the WAP Proxy/gateway typically uses DNS name resolution to determine the IP address of the server on which the document resides.

The client device would first identify a proxy/gateway that is reachable through Bluetooth, then it would use the service discovery protocol to present the user with a server name or description. When the user selects a server, then the WAP client downloads the home page of the server (as determined by the discovery process; see section 4.1.1.1 on page 507) Once the user has navigated to the home page of the desired server, then all subsequent URLs are relative to this home page. This scenario presumes that the WAP Proxy/gateway and WAP Content server are all co-located in the Bluetooth device, although this structure is not required for interoperability.

A WAP Proxy/gateway/Server will typically provide a default URL containing the home page content for the server. A proxy-only device typically provides no URL or associated content.

4.3 NETWORK SUPPORT FOR WAP

The following specifies a protocol stack, which may be used below the WAP components. Support for other protocol stack configurations is optional, and must be indicated through the Bluetooth Service Discovery Protocol.

4.3.1 PPP/RFCOMM

Devices that support Bluetooth as a bearer for WAP services using PPP provide the following protocol stack support:

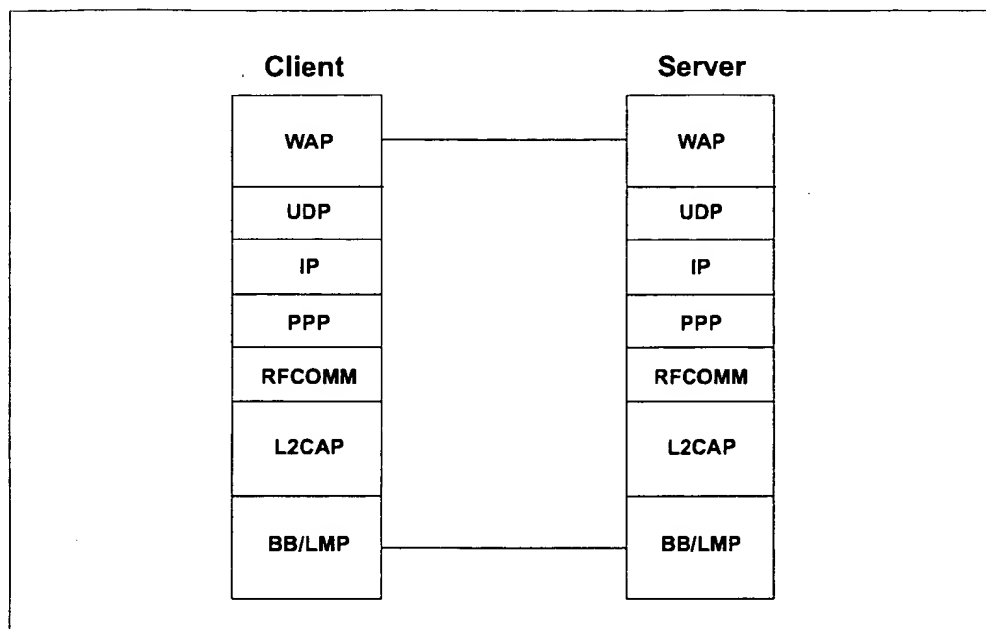


Figure 4.2: Protocol Support for WAP

For the purposes of interoperability, this document assumes that a WAP client conforms to the role of Data Terminal as defined in LAN Access Profile using PPP [6]. Additionally, the WAP server or proxy device is assumed to conform to the role of the LAN Access Point defined in [6].

The Baseband (page 33), LMP (page 185) and L2CAP (page 245) are the OSI layer 1 and 2 Bluetooth protocols. RFCOMM (page 385) is the Bluetooth adaptation of GSM TS 07.10 [1]. SDP (page 323) is the Bluetooth Service Discovery Protocol.

PPP is the IETF Point-to-Point Protocol [3]. WAP is the Wireless Application Protocol stack and application environment [5].

5 INTEROPERABILITY REQUIREMENTS

5.1 STAGE 1 – BASIC INTEROPERABILITY

Stage 1 interoperability for WAP over Bluetooth (all mandatory):

- Provide WAP Class A device compliance [7]
- Provide, through service discovery mechanisms, the network address for devices that support WAP proxy/gateway functionality.

5.2 STAGE 2 – ADVANCED INTEROPERABILITY

Stage 2 interoperability for WAP over Bluetooth (mandatory):

- All Stage 1 interoperability requirements are supported
- Provide Server Name and information about Server/Proxy capabilities through service discovery.
- Provide Client Name and information about Client Capabilities through service discovery.
- Asynchronous Notifications for Server.
- Asynchronous Notifications for Client.

6 SERVICE DISCOVERY

6.1 SDP SERVICE RECORDS

Service records are provided as a mechanism through which WAP client devices and proxy/gateways become aware of each other dynamically. This usage differs from other WAP bearers in that the relationship between the two devices will be transitory. That is, a Bluetooth device will not have a bearer-specific address configured or provisioned to a specific proxy/gateway.

Clients and proxy/gateways become aware of each other as they come in proximity of one another. The Bluetooth Service Discovery Protocol allows the devices to query the capabilities of each other as listed in the Interoperability Requirements section of this document.

Table 6.1 shows the service record for the WAP Proxy/gateway device.

Item	Definition	Type	Value	AttrID	Req
ServiceClassIDList				0x0001	M
ServiceClass0	WAP Proxy/Gateway	UUID	WAP		M
BluetoothProfileDescriptorList					M
ProfileDescriptor0				0x0009	M
Profile	Supported Profile	UUID	LANAccess UsingPPP [4]		M
Version	Profile Version	Uint16	(varies)		M
ProtocolDescriptorList					O
Descriptor0	UDP	UUID	UDP		O
Parameter0	WSP Connectionless Session Port No.	Uint16	9200 (default)		O
Parameter1	WTP Session Port No.	Uint16	9201 (default)		O
Parameter2	WSP Secure Connectionless Port No.	Uint16	9202 (default)		O
Parameter3	WTP Secure Session Port No.	Uint16	9203 (default)		O
Parameter4	WAP vCard Port No.	Uint16	9204 (default)		O
Parameter5	WAP vCal Port No.	Uint16	9205 (default)		O
Parameter6	WAP vCard Secure Port No.	Uint16	9206 (default)		O
Parameter7	WAP vCal Secure Port No.	Uint16	9207 (default)		O

Table 6.1: Service Record format for WAP Proxy/Gateway devices

Interoperability Requirements for Bluetooth as a WAP Bearer

Bluetooth

Item	Definition	Type	Value	AttrID	Req
ServiceNam	Displayable Text name	String	(varies, e.g. 'Airport infor- mation')		
NetworkAddress	IP Network Address of Server	UInt32	(varies)		M
WAPGateway*	Indicates if device is origin server or proxy	UInt8	0x01 = Origin Server; 0x02 = Proxy; 0x03 = Origin Server and Proxy		M
HomePageURL	URL of home page document	URL			C1†

Table 6.1: Service Record format for WAP Proxy/Gateway devices

*. Stage 2 interoperability requirements.

†. If this parameter is omitted, then a default is assumed for origin servers as:
<http://networkaddress/index.wml>

Item	Definition	Type	Value	AttrID	Req
ServiceClassIDList				0x0001	M
ServiceClass0	WAP Client	UUID	WAP_CLIENT		M
BluetoothProfile DescriptorList					M
ProfileDescriptor0				0x0009	M
Profile	Supported Profile	UUID	LANAccess UsingPPP [4]		M
Version	Profile Version	UInt16	(varies)		M
ServiceName	Displayable Text name of client	String	(varies)		O

Table 6.2: Service Record format for WAP Client devices

6.2 SDP PROTOCOL DATA UNITS

Table 6.3 shows the specified SDP PDUs (Protocol Data Units), which are required for WAP Interoperability.

PDU No.	SDP PDU	Ability to Send		Ability to Retrieve	
		WAP Client	WAP Proxy	WAP Client	WAP Proxy
1	SdpErrorResponse	M	M	M	M
2	SdpServiceSearchAttributeRequest	M	O	M	M
3	SdpServiceSearchAttributeResponse	M	M	M	M

Table 6.3: SDP PDU:s

6.3 SERVICE DISCOVERY PROCEDURE

In the simplest form, the signaling can be like this:

WAP Client or Proxy		WAP Client or Proxy
	SdpServiceSearchAttributeRequest =====>	
	SdpServiceSearchAttributeResponse <=====	

WAP service discovery procedures are symmetrical. Each device must be able to handle all of the PDUs without regard for the current device role. A minimal implementation must return the service name string.

7 REFERENCES

- [1] TS 101 369 (GSM 07.10) version 6.2.0
- [2] Simpson, W., Editor, "The Point-to-Point Protocol (PPP)", STD 50, RFC 1661, Daydreamer, July 1994.
- [3] Simpson, W., Editor, "PPP in HDLC Framing", STD 51, RFC 1662, Daydreamer, July 1994.
- [4] See Appendix VIII, "Bluetooth Assigned Numbers" on page 1009
- [5] Wireless Application Protocol Forum, "Wireless Application Protocol", version 1.0, 1998
- [6] Bluetooth Special Interest Group, "Bluetooth LAN Access Profile using PPP"
- [7] Wireless Application Protocol Forum, "WAP Conformance", Draft version 27 May 1998



Part H:1



HOST CONTROLLER INTERFACE FUNCTIONAL SPECIFICATION

This document describes the functional specification for the Host Controller Interface (HCI). The HCI provides a command interface to the baseband controller and link manager, and access to hardware status and control registers. This interface provides a uniform method of accessing the Bluetooth baseband capabilities.

CONTENTS

1	Introduction	524
1.1	Lower Layers of the Bluetooth Software Stack	524
1.2	Bluetooth Hardware Block Diagram	525
1.2.1	Link Controller	526
1.2.2	CPU Core	526
1.3	Possible Physical Bus Architectures	527
1.3.1	USB HCI Architecture	527
1.3.2	PC Card HCI Architecture	527
2	Overview of Host Controller Transport Layer	528
3	HCI Flow Control	529
4	HCI Commands	531
4.1	Introduction	531
4.2	Terminology	531
4.3	Data and Parameter Formats	532
4.4	Exchange of HCI-Specific Information	532
4.4.1	HCI Command Packet	532
4.4.2	HCI Event Packet	535
4.4.3	HCI Data Packets	536
4.5	Link Control Commands	540
4.5.1	Inquiry	542
4.5.2	Inquiry_Cancel	544
4.5.3	Periodic_Inquiry_Mode	545
4.5.4	Exit_Periodic_Inquiry_Mode	548
4.5.5	Create_Connection	549
4.5.6	Disconnect	552
4.5.7	Add_SCO_Connection	553
4.5.8	Accept_Connection_Request	555
4.5.9	Reject_Connection_Request	557
4.5.10	Link_Key_Request_Reply	558
4.5.11	Link_Key_Request_Negative_Reply	560
4.5.12	PIN_Code_Request_Reply	561
4.5.13	PIN_Code_Request_Negative_Reply	563
4.5.14	Change_Connection_Packet_Type	564
4.5.15	Authentication_Requested	567
4.5.16	Set_Connection_Encryption	568
4.5.17	Change_Connection_Link_Key	569
4.5.18	Master_Link_Key	570

4.5.19	Remote_Name_Request.....	571
4.5.20	Read_Remote_Supported_Features.....	573
4.5.21	Read_Remote_Version_Information.....	574
4.5.22	Read_Clock_Offset.....	575
4.6	Link Policy Commands.....	576
4.6.1	Hold_Mode.....	578
4.6.2	Sniff_Mode.....	580
4.6.3	Exit_Sniff_Mode.....	582
4.6.4	Park_Mode.....	583
4.6.5	Exit_Park_Mode.....	585
4.6.6	QoS_Setup.....	586
4.6.7	Role_Discovery.....	588
4.6.8	Switch_Role.....	589
4.6.9	Read_Link_Policy_Settings.....	590
4.6.10	Write_Link_Policy_Settings.....	592
4.7	Host Controller & Baseband Commands.....	594
4.7.1	Set_Event_Mask.....	600
4.7.2	Reset.....	602
4.7.3	Set_Event_Filter.....	603
4.7.4	Flush.....	609
4.7.5	Read_PIN_Type.....	611
4.7.6	Write_PIN_Type.....	612
4.7.7	Create_New_Unit_Key.....	613
4.7.8	Read_Stored_Link_Key.....	614
4.7.9	Write_Stored_Link_Key.....	616
4.7.10	Delete_Stored_Link_Key.....	618
4.7.11	Change_Local_Name.....	620
4.7.12	Read_Local_Name.....	621
4.7.13	Read_Connection_Accept_Timeout.....	622
4.7.14	Write_Connection_Accept_Timeout.....	623
4.7.15	Read_Page_Timeout.....	624
4.7.16	Write_Page_Timeout.....	625
4.7.17	Read_Scan_Enable.....	626
4.7.18	Write_Scan_Enable.....	627
4.7.19	Read_Page_Scan_Activity.....	628
4.7.20	Write_Page_Scan_Activity.....	630
4.7.21	Read_Inquiry_Scan_Activity.....	632
4.7.22	Write_Inquiry_Scan_Activity.....	634

4.7.23	Read_Authentication_Enable	636
4.7.24	Write_Authentication_Enable	637
4.7.25	Read_Encryption_Mode	638
4.7.26	Write_Encryption_Mode	639
4.7.27	Read_Class_of_Device	641
4.7.28	Write_Class_of_Device	642
4.7.29	Read_Voice_Setting	643
4.7.30	Write_Voice_Setting	645
4.7.31	Read_Automatic_Flush_Timeout	647
4.7.32	Write_Automatic_Flush_Timeout	649
4.7.33	Read_Num_Broadcast_Retransmissions	651
4.7.34	Write_Num_Broadcast_Retransmissions	652
4.7.35	Read_Hold_Mode_Activity	653
4.7.36	Write_Hold_Mode_Activity	655
4.7.37	Read_Transmit_Power_Level	656
4.7.38	Read_SCO_Flow_Control_Enable	658
4.7.39	Write_SCO_Flow_Control_Enable	659
4.7.40	Set_Host_Controller_To_Host_Flow_Control	660
4.7.41	Host_Buffer_Size	661
4.7.42	Host_Number_Of_Completed_Packets	663
4.7.43	Read_Link_Supervision_Timeout	665
4.7.44	Write_Link_Supervision_Timeout	667
4.7.45	Read_Number_Of_Supported_IAC	669
4.7.46	Read_Current_IAC_LAP	670
4.7.47	Write_Current_IAC_LAP	671
4.7.48	Read_Page_Scan_Period_Mode	673
4.7.49	Write_Page_Scan_Period_Mode	675
4.7.50	Read_Page_Scan_Mode	676
4.7.51	Write_Page_Scan_Mode	677
4.8	Informational Parameters	678
4.8.1	Read_Local_Version_Information	679
4.8.2	Read_Local_Supported_Features	681
4.8.3	Read_Buffer_Size	682
4.8.4	Read_Country_Code	684
4.8.5	Read_BD_ADDR	685
4.9	Status Parameters	686
4.9.1	Read_Failed_Contact_Counter	687
4.9.2	Reset_Failed_Contact_Counter	689

4.9.3	Get_Link_Quality	691
4.9.4	Read_RSSI	693
4.10	Testing Commands	695
4.10.1	Read_Loopback_Mode	696
4.10.2	Write_Loopback_Mode	699
4.10.3	Enable_Device_Under_Test_Mode.....	702
5	Events	703
5.1	Event.....	703
5.2	Possible Events	706
5.2.1	Inquiry Complete event	706
5.2.2	Inquiry Result event	707
5.2.3	Connection Complete event.....	709
5.2.4	Connection Request event.....	711
5.2.5	Disconnection Complete event	712
5.2.6	Authentication Complete event	713
5.2.7	Remote Name Request Complete event	714
5.2.8	Encryption Change event.....	715
5.2.9	Change Connection Link Key Complete event	716
5.2.10	Master Link Key Complete event	717
5.2.11	Read Remote Supported Features Complete event...	718
5.2.12	Read Remote Version Information Complete event....	719
5.2.13	QoS Setup Complete event	721
5.2.14	Command Complete event	723
5.2.15	Command Status event.....	724
5.2.16	Hardware Error event.....	725
5.2.17	Flush Occurred event.....	726
5.2.18	Role Change event	727
5.2.19	Number Of Completed Packets event.....	728
5.2.20	Mode Change event.....	730
5.2.21	Return Link Keys event.....	732
5.2.22	PIN Code Request event	733
5.2.23	Link Key Request event	734
5.2.24	Link Key Notification event.....	735
5.2.25	Loopback Command event	736
5.2.26	Data Buffer Overflow event.....	737
5.2.27	Max Slots Change event.....	738
5.2.28	Read Clock Offset Complete event.....	739
5.2.29	Connection Packet Type Changed event.....	740

5.2.30	QoS Violation event.....	742
5.2.31	Page Scan Mode Change event	743
5.2.32	Page Scan Repetition Mode Change event	744
6	List of Error Codes.....	745
6.1	List of Error Codes	745
6.2	HCI Error Code Usage Descriptions	747
6.3	Unknown HCI Command (0x01)	747
6.4	No Connection (0x02)	748
6.5	Hardware Failure (0x03)	748
6.6	Page Timeout (0x04).....	748
6.7	Authentication Failed (0x05)	748
6.8	Key Missing (0x06).....	748
6.9	Memory Full (0x07)	749
6.10	Connection Timeout (0x08).....	749
6.11	Max Number Of Connections (0x09).....	749
6.12	Max Number Of SCO Connections To A Device (0x0A)	750
6.13	ACL Connection Already Exists (0x0B).....	750
6.14	Command Disallowed (0x0C)	750
6.15	Host Rejected due to ... (0x0D-0x0F).....	750
6.16	Host Timeout (0x10).....	751
6.17	Unsupported Feature or Parameter Value (0x11)	751
6.18	Invalid HCI Command Parameters (0x12)	751
6.19	Other End Terminated Connection: ... (0x13-0x15).....	752
6.20	Connection Terminated By Local Host (0x16).....	752
6.21	Repeated Attempts (0x17)	752
6.22	Pairing Not Allowed (0x18).....	753
6.23	Unsupported Remote Feature (0x1A).....	753
6.24	Unspecified error (0x1F)	753
6.25	Unsupported LMP Parameter Value (0x20)	753
6.26	Role Change Not Allowed (0x21).....	753
6.27	LMP Response Timeout (0x22)	754
6.28	LMP Error Transaction Collision (0x23).....	754
6.29	LMP PDU Not Allowed (0x24)	754
7	List of Acronyms and Abbreviations.....	755
8	List of Figures.....	756
9	List of Tables	757

1 INTRODUCTION

This document describes the functional specifications for the Host Controller Interface (HCI). The HCI provides a uniform interface method of accessing the Bluetooth hardware capabilities. The next two sections provide a brief overview of the lower layers of the Bluetooth software stack and of the Bluetooth hardware. Section 2, provides an overview of the Lower HCI Device Driver Interface on the host device. Section 3, describes the flow control used between the Host and the Host Controller. Section 4, describes each of the HCI Commands in details, identifies parameters for each of the commands, and lists events associated with each command.

1.1 LOWER LAYERS OF THE BLUETOOTH SOFTWARE STACK

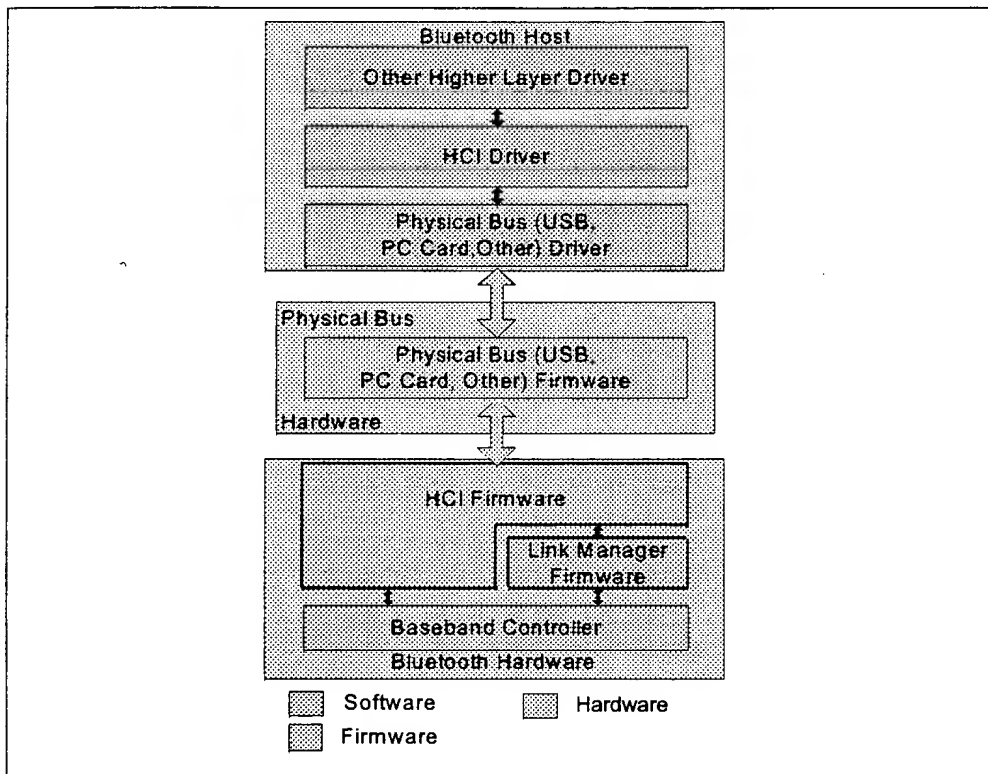


Figure 1.1: Overview of the Lower Software Layers

Figure 1.1, provides an overview of the lower software layers. The HCI firmware implements the HCI Commands for the Bluetooth hardware by accessing baseband commands link manager commands, hardware status registers, control registers, and event registers.

Several layers may exist between the HCI driver on the host system and the HCI firmware in the Bluetooth hardware. These intermediate layers, the Host Controller Transport Layer, provide the ability to transfer data without intimate knowledge of the data.

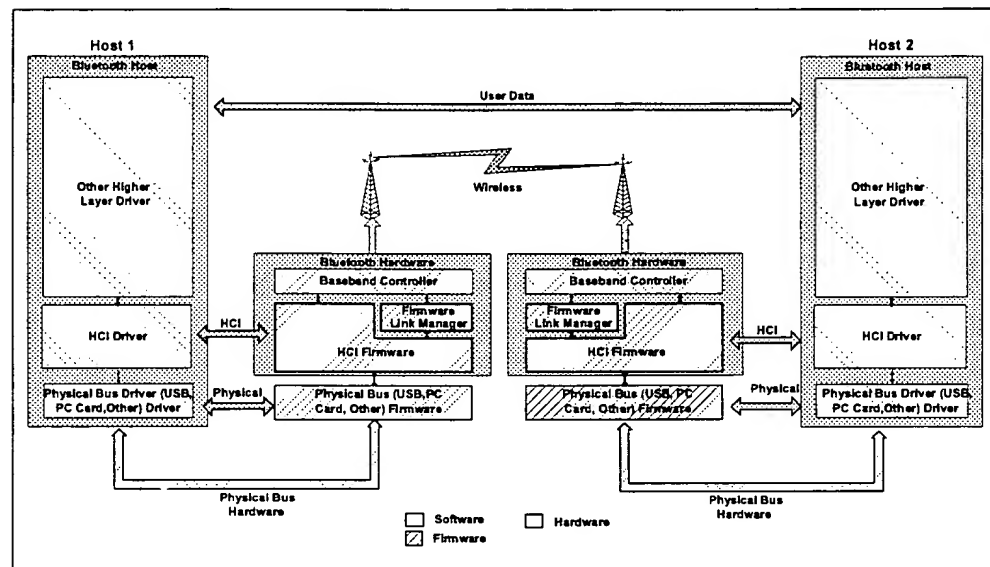


Figure 1.2: End to End Overview of Lower Software Layers to Transfer Data

Figure 1.2, illustrates the path of a data transfer from one device to another. The HCI driver on the Host exchanges data and commands with the HCI firmware on the Bluetooth hardware. The Host Control Transport Layer (i.e. physical bus) driver provides both HCI layers with the ability to exchange information with each other.

The Host will receive asynchronous notifications of HCI events independent of which Host Controller Transport Layer is used. HCI events are used for notifying the Host when something occurs. When the Host discovers that an event has occurred it will then parse the received event packet to determine which event occurred.

1.2 BLUETOOTH HARDWARE BLOCK DIAGRAM

A general overview of the Bluetooth hardware is outlined in Figure 1.3 on page 526. It consists of an analog part – the Bluetooth radio, and a digital part – the Host Controller. The Host Controller has a hardware digital signal processing part – the Link Controller (LC), a CPU core, and it interfaces to the host environment. The hardware and software parts of the Host Controller are described below.

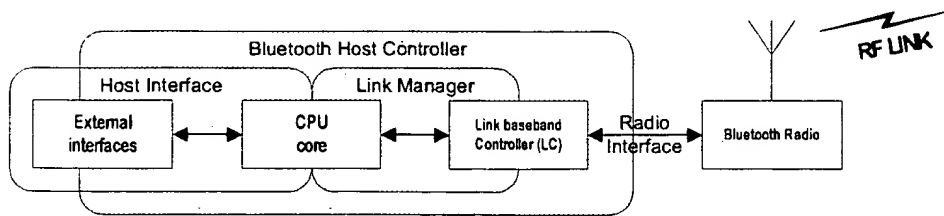


Figure 1.3: Bluetooth Hardware Architecture Overview.

1.2.1 Link Controller

The Link Controller (LC) consists of hardware and software parts that perform Bluetooth baseband processing, and physical layer protocols such as ARQ-protocol and FEC coding.

The functions performed by the Link Controller include:

- Transfer types with selected Quality-of-Service (QoS) parameters
- Asynchronous transfers with guaranteed delivery using hardware fast Automatic Repeat reQuest (fARQ). Frames can be flushed from the retransmission buffer, for use with isochronous data
- Synchronous transfers
- Audio coding. A power-efficient hardware implementation of a robust 64 Kbits/s Continuous Variable Slope Delta (CVSD) coding, as well as 64 Kbits/s log-PCM
- Encryption

1.2.2 CPU Core

The CPU core will allow the Bluetooth module to handle Inquiries and filter Page requests without involving the host device. The Host Controller can be programmed to answer certain Page messages and authenticate remote links.

The Link Manager (LM) software runs on the CPU Core. The LM discovers other remote LMs and communicates with them via the Link Manager Protocol (LMP) to perform its service provider role using the services of the underlying Link Controller (LC). For details see "Link Manager Protocol" on page 185

1.3 POSSIBLE PHYSICAL BUS ARCHITECTURES

Bluetooth devices will have various physical bus interfaces that could be used to connect to the Bluetooth hardware. These buses may have different architectures and different parameters. The Bluetooth Host Controller will initially support two physical bus architectures, USB, and PC Card.

1.3.1 USB HCI Architecture

The following block diagram shows the Bluetooth connection to the Host PC via the USB HCI. USB can handle several logic channels over the same single physical channel (via Endpoints). Therefore control, data, and voice channels do not require any additional physical interfaces. Note that there is no direct access to registers/memory on the Bluetooth module over USB. Instead, this is done by using the appropriate HCI Commands and by using the Host Controller Transport Layer interface.

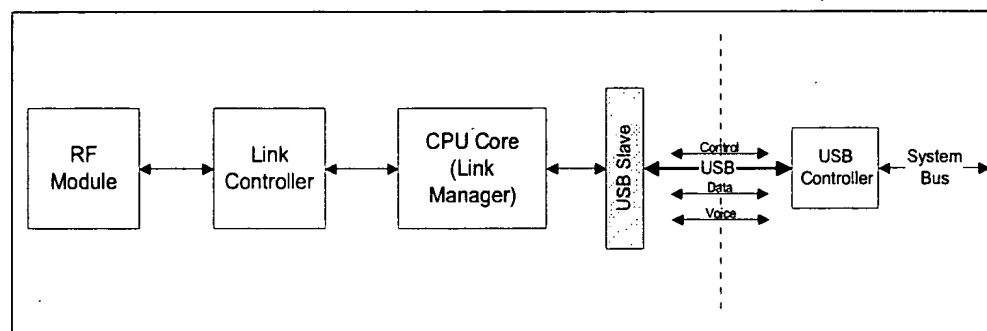


Figure 1.4: Bluetooth Block Diagram with USB HCI

1.3.2 PC Card HCI Architecture

Besides the USB interface, derivatives of the ISA bus (Compact Flash/PC Card interfaces) are an option for an integrated PC solution. Unlike USB, all traffic between the Host and the Bluetooth module will go across the PC Card bus interface. Communications between the host PC and the Bluetooth module will be primarily done directly via registers/memory. The following block diagram shows the data flow for a PC-Card HCI.

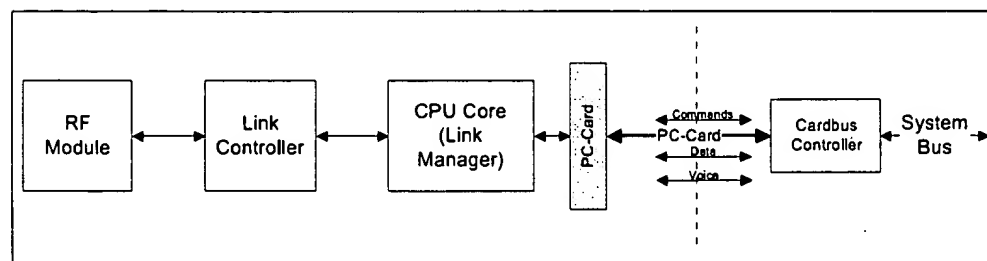


Figure 1.5: Bluetooth Block Diagram with PC-Card HCI

2 OVERVIEW OF HOST CONTROLLER TRANSPORT LAYER

The host driver stack has a transport layer between the Host Controller driver and the Host Controller. On a laptop, this transport layer might be PC Card or Universal Serial Bus (USB).

The main goal of this transport layer is transparency. The Host Controller driver (which talks to the Host Controller) should not care whether it is running over USB or a PC Card. Nor should USB or PC Card require any visibility into the data that the Host Controller driver passes to the Host Controller. This allows the interface (HCI) or the Host Controller to be upgraded without affecting the transport layer.

The Host Controller Transport Layer is described in separate documents for each physical media.

- "HCI USB Transport Layer" on page 759.
- "HCI RS232 Transport Layer" on page 775.
- "HCI UART Transport Layer" on page 795.

3 HCI FLOW CONTROL

Flow control is used in the direction from the Host to the Host Controller to avoid filling up the Host Controller data buffers with ACL data destined for a remote device (connection handle) that is not responding. It is the Host that manages the data buffers of the Host Controller.

On Initialization, the Host will issue the `Read_Buffer_Size` command. Two of the return parameters of this command determine the maximum size of HCI ACL and SCO Data Packets (excluding header) sent from the Host to the Host Controller. There are also two additional return parameters that specify the total number of HCI ACL and SCO Data Packets that the Host Controller can have waiting for transmission in its buffers. When there is at least one connection to another device, or when in local loopback mode, the Host Controller uses the `Number Of Completed Packets` event to control the flow of data from the Host. This event contains a list of connection handles and a corresponding number of HCI Data Packets that have been completed (transmitted, flushed, or looped back to the Host) since the previous time the event was returned (or since the connection was established, if the event has not been returned before for a particular connection handle). Based on the information returned in this event, and the return parameters of the `Read_Buffer_Size` command that specify the total number of HCI ACL and SCO Data Packets that can be stored in the Host Controller, the Host can decide for which Connection Handles the following HCI Data Packets should be sent. After every time it has sent an HCI Data Packet, the Host must assume that the free buffer space for the corresponding link type (ACL or SCO) in the Host Controller has decreased by one HCI Data Packet. When the Host receives a new `Number Of Completed Packets` event, the Host gets information about how much the buffer usage has decreased since the previous time the event was returned. It can then calculate the actual current buffer usage. While the Host Controller has HCI data packets in its buffer, it must keep sending the `Number Of Completed Packets` event to the Host at least periodically, until it finally reports that all the pending ACL Data Packets have been transmitted or flushed. The rate with which this event is sent is manufacturer specific. Note that `Number Of Completed Packets` events will not report on SCO connection handles if SCO Flow Control is disabled. (See `Read/Write_SCO_Flow_Control_Enable` on page 658 and page 659.)

Note that for each individual Connection Handle, the data must be sent to the Host Controller in HCI Data Packets in the order in which it was created in the Host. The Host Controller must also transmit data on the air that is received from the Host for a given Connection Handle in the same order as it is received from the Host. Furthermore, data that is received on the air from another device must, for the corresponding Connection Handle, be sent in HCI Data Packets to the Host in the same order as it is received. This means that the scheduling is made on a Connection Handle basis. For each individual Connection Handle, the order of the data must not be changed from the order in which the data has been created.

In certain cases, flow control may also be necessary in the direction from the Host Controller to the Host. There is therefore a command – `Set_Host_Controller_To_Host_Flow_Control` – to turn flow control on or off in that direction. If turned on, it works in exactly the same way as described above. On initialization, the Host uses the `Host_Buffer_Size` command to notify the Host Controller about the maximum size of HCI ACL and SCO Data Packets sent from the Host Controller to the Host. The command also contains two additional command parameters to notify the Host Controller about the total number of ACL and SCO Data Packets that can be stored in the data buffers of the Host. The Host then uses the `Host_Number_Of_Completed_Packets` command in exactly the same way as the Host Controller uses the Number Of Completed Packets event (as was previously described in this section). The `Host_Number_Of_Completed_Packets` command is a special command for which no command flow control is used, and which can be sent anytime there is a connection or when in local loopback mode. This makes it possible for the flow control to work in exactly the same way in both directions, and the flow of normal commands will not be disturbed.

When the Host receives a Disconnection Complete event, the Host can assume that all HCI Data Packets that have been sent to the Host Controller for the returned `Connection_Handle` have been flushed, and that the corresponding data buffers have been freed. The Host Controller does not have to notify the Host about this in a Number Of Completed Packets event. If flow control is also enabled in the direction from the Host Controller to the Host, the Host Controller can after it has sent a Disconnection Complete event assume that the Host will flush its data buffers for the sent `Connection_Handle` when it receives the Disconnection Complete event. The Host does not have to notify the Host Controller about this in a `Host_Number_Of_Completed_Packets` command.

4 HCI COMMANDS

4.1 INTRODUCTION

The HCI provides a uniform command method of accessing the Bluetooth hardware capabilities. The HCI Link commands provide the Host with the ability to control the link layer connections to other Bluetooth devices. These commands typically involve the Link Manager (LM) to exchange LMP commands with remote Bluetooth devices. For details see "Link Manager Protocol" on page 185.

The HCI Policy commands are used to affect the behavior of the local and remote LM. These Policy commands provide the Host with methods of influencing how the LM manages the piconet. The Host Controller & Baseband, Informational, and Status commands provide the Host access to various registers in the Host Controller.

HCI commands may take different amounts of time to be completed. Therefore, the results of commands will be reported back to the Host in the form of an event. For example, for most HCI commands the Host Controller will generate the Command Complete event when a command is completed. This event contains the return parameters for the completed HCI command. To detect errors on the HCI-Transport Layer a response timeout needs to be defined between the Host Controller receiving a command and sending a response to the command (e.g. a Command Complete or Command Status event). Since the maximum response timeout is strongly dependent on the HCI-Transport Layer used, it is recommended to use a default value of one second for this timer. This amount of time is also dependent on the number of commands unprocessed in the command queue.

4.2 TERMINOLOGY

Baseband Packet: The smallest unit of data that is transmitted by one device to another, as defined by the "Baseband Specification" on page 33.

Packet: A higher-level protocol message than the baseband packet, currently only L2CAP (see "Logical Link Control and Adaptation Protocol Specification" on page 245) is defined, but additional packet types may be defined later.

Connection Handle: A connection handle is a 12-bit identifier which is used to uniquely address a data/voice connection from one Bluetooth device to another. The connection handles can be visualized as identifying a unique data pipe that connects two Bluetooth devices. The connection handle is maintained for the lifetime of a connection, including when a device enters Park, Sniff, or Hold mode. The Connection Handle value has local scope between Host and Host Controller. There can be multiple connection handles for any given pair of Bluetooth devices but only one ACL connection.

Event: A mechanism that the HCI uses to notify the Host for command completion, link layer status changes, etc.

4.3 DATA AND PARAMETER FORMATS

- All values are in Binary and Hexadecimal Little Endian formats unless otherwise noted
- In addition, all parameters which can have negative values must use 2's complement when specifying values
- Arrayed parameters are specified using the following notation: ParameterA[i]. If more than one set of arrayed parameters are specified (e.g. ParameterA[i], ParameterB[i]), then the order of the parameters are as follows: ParameterA[0], ParameterB[0], ParameterA[1], ParameterB[1], ParameterA[2], ParameterB[2], ... ParameterA[n], ParameterB[n]
- Unless noted otherwise, all parameter values are sent and received in Little Endian format (i.e. for multi-byte parameters the rightmost (Least Signification Byte) is transmitted first)
- All command and event parameters that are not-arrayed and all elements in an arrayed parameter have fixed sizes (an integer number of bytes). The parameters and the size of each not arrayed parameter (or of each element in an arrayed parameter) contained in a command or an event is specified for each command or event. The number of elements in an arrayed parameter is not fixed.

4.4 EXCHANGE OF HCI-SPECIFIC INFORMATION

The Host Controller Transport Layer provides transparent exchange of HCI-specific information. These transporting mechanisms provide the ability for the Host to send HCI commands, ACL data, and SCO data to the Host Controller. These transport mechanisms also provide the ability for the Host to receive HCI events, ACL data, and SCO data from the Host Controller.

Since the Host Controller Transport Layer provides transparent exchange of HCI-specific information, the HCI specification specifies the format of the commands, events, and data exchange between the Host and the Host Controller. The next sections specify the HCI packet formats.

4.4.1 HCI Command Packet

The HCI Command Packet is used to send commands to the Host Controller from the Host. The format of the HCI Command Packet is shown in Figure 4.1, and the definition of each field is explained below. When the Host Controller completes most of the commands, a Command Complete event is sent to the Host. Some commands do not receive a Command Complete event when they have been completed. Instead, when the Host Controller receives one of these commands the Host Controller sends a Command Status event back to the Host when it has begun to execute the command. Later on, when the actions associated with the command have finished, an event that is associated with the sent command will be sent by the Host Controller to the Host. However, if the command does not begin to execute (there may be a parameter error or the

command may currently not be allowed), the event associated with the sent command will not be returned. The Command Status event will, in this case, return the appropriate error code in the Status parameter. On initial power-on, and after a reset, the Host can send a maximum of one outstanding HCI Command Packet until a Command Complete or Command Status event has been received. If an error occurs for a command for which a Command Complete event is returned, the Return_Parameters field may not contain all the return parameters specified for the command. The Status parameter, which explains the error reason and which is the first return parameter, will always be returned. If there is a Connection_Handle parameter or a BD_ADDR parameter right after the Status parameter, this parameter will also be returned so that the Host can identify to which instance of a command the Command Complete event belongs. In this case, the Connection_Handle or BD_ADDR parameter will have exactly the same value as that in the corresponding command parameter. It is implementation specific whether more parameters will be returned in case of an error.

Note: The BD_ADDR return parameter of the command Read_BD_ADDR is not used to identify to which instance of the Read_BD_ADDR command the Command Complete event belongs. It is therefore not mandatory for the Host Controller to return this parameter in case of an error.

If an error occurs for a command for which no Command Complete event is returned, all parameters returned with the event associated with this command may not be valid. The Host must take care as to which parameters may have valid values depending on the value of the Status parameter of the Complete event associated with the given command. The Command Complete and Command Status events contain a parameter called Num_HCI_Command_Packets, which indicates the number of HCI Command Packets the Host is currently allowed to send to the Host Controller. The Host Controller may buffer one or more HCI command packets, but the Host Controller must start performing the commands in the order in which they are received. The Host Controller can start performing a command before it completes previous commands. Therefore, the commands do not always complete in the order they are started. The Host Controller must be able to accept HCI Command Packets with up to 255 bytes of data excluding the HCI Command Packet header.

Each command is assigned a 2 byte Opcode used to uniquely identify different types of commands. The Opcode parameter is divided into two fields, called the OpCode Group Field (OGF) and OpCode Command Field (OCF). The OGF occupies the upper 6 bits of the Opcode, while the OCF occupies the remaining 10 bits. The OGF of 0x3F is reserved for vendor-specific debug commands. The OGF of 0x3E is reserved for Bluetooth Logo Testing. The organization of the Opcodes allows additional information to be inferred without fully decoding the entire Opcode.

Note: the OGF composed of all 'ones' has been reserved for vendor-specific debug commands. These commands are vendor-specific and are used during manufacturing, for a possible method for updating firmware, and for debugging.

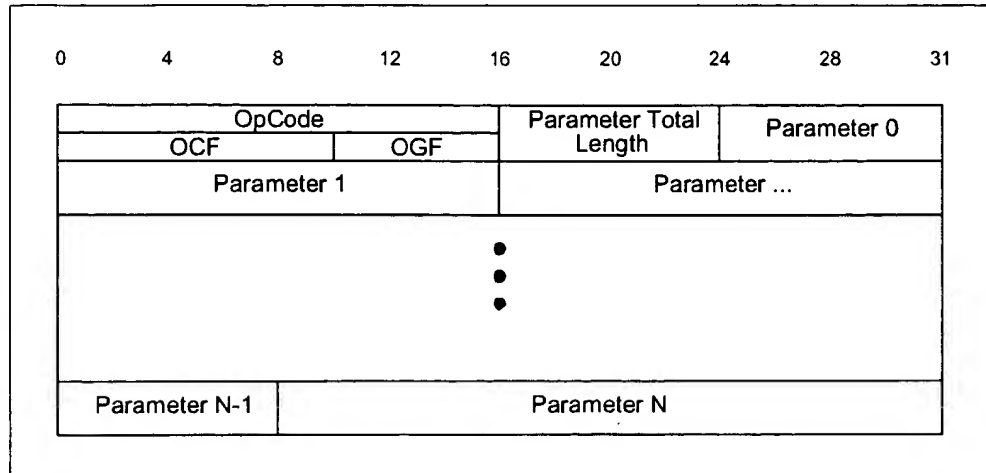


Figure 4.1: HCI Command Packet

Op_Code:

Size: 2 Bytes

Value	Parameter Description
0xFFFF	OGF Range (6 bits): 0x00-0x3F (0x3E reserved for Bluetooth logo testing and 0x3F reserved for vendor-specific debug commands) OCF Range (10 bits): 0x0000-0x03FF

Parameter_Total_Length:

Size: 1 Byte

Value	Parameter Description
0xFF	Lengths of all of the parameters contained in this packet measured in bytes. (N.B.: total length of parameters, <u>not</u> number of parameters)

Parameter 0 - N:

Size: Parameter Total Length

Value	Parameter Description
0xFF	Each command has a specific number of parameters associated with it. These parameters and the size of each of the parameters are defined for each command. Each parameter is an integer number of bytes in size.

4.4.2 HCI Event Packet

The HCI Event Packet is used by the Host Controller to notify the Host when events occur. The Host must be able to accept HCI Event Packets with up to 255 bytes of data excluding the HCI Event Packet header. The format of the HCI Event Packet is shown in Figure 4.2, and the definition of each field is explained below.

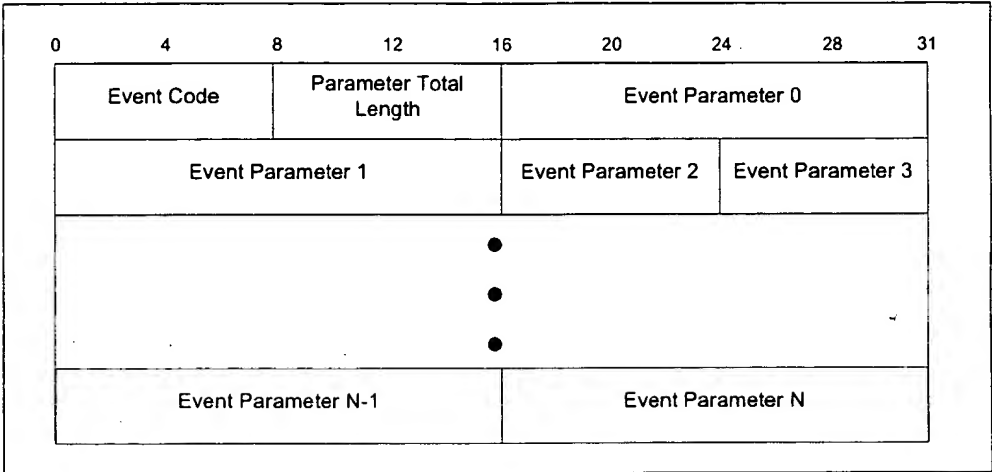


Figure 4.2: HCI Event Packet

Event_Code: Size: 1 Byte

Value	Parameter Description
0xXX	Each event is assigned a 1-Byte event code used to uniquely identify different types of events. Range: 0x00-0xFF (The event code 0xFF is reserved for the event code used for vendor-specific debug events. In addition, the event code 0xFE is also reserved for Bluetooth Logo Testing)

Parameter_Total_Length: Size: 1 Byte

Value	Parameter Description
0xXX	Length of all of the parameters contained in this packet, measured in bytes

Event_Parameter 0 - N: Size: Parameter Total Length

Value	Parameter Description
0xXX	Each event has a specific number of parameters associated with it. These parameters and the size of each of the parameters are defined for each event. Each parameter is an integer number of bytes in size.

4.4.3 HCI Data Packets

HCI Data Packets are used to exchange data between the Host and Host Controller. The data packets are defined for both ACL and SCO data types. The format of the HCI ACL Data Packet is shown in Figure 4.3, and the format of the SCO Data Packet is shown in Figure 4.4. The definition for each of the fields in the data packets is explained below.

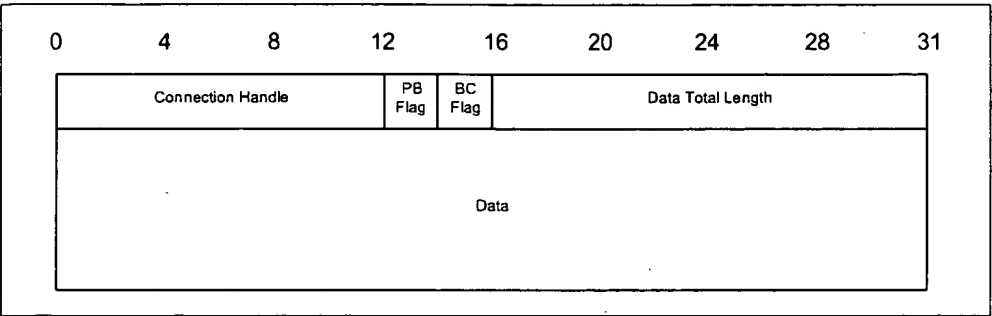


Figure 4.3: HCI ACL Data Packet

Connection_Handle:**Size: 12 Bits**

Value	Parameter Description
0xXXX	<p>Connection Handle to be used for transmitting a data packet or segment. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)</p> <p>The first time the Host sends an HCI Data Packet with Broadcast_Flag set to 01b (active broadcast) or 10b (piconet broadcast) after a power-on or a reset, the value of the Connection_Handle parameter must be a value which is not currently assigned by the Host Controller. The Host must use different connection handles for active broadcast and piconet broadcast. The Host Controller must then continue to use the same connection handles for each type of broadcast until a reset is made.</p> <p>Note: The Host Controller must not send a Connection Complete event containing a new Connection_Handle that it knows is used for broadcast. Note: In some situations, it may happen that the Host Controller sends a Connection Complete event before having interpreted a Broadcast packet received from the Host, and that the Connection_Handles of both Connection Complete event and HCI Data packet are the same. This conflict has to be avoided as follows:</p> <p>If a Connection Complete event is received containing one of the connection handles used for broadcast, the Host has to wait before sending any packets for the new connection until it receives a Number Of Completed Packets event indicating that there are no pending broadcast packets belonging to the connection handle. In addition, the Host must change the Connection_Handle used for the corresponding type of broadcast to a Connection_Handle which is currently not assigned by the Host Controller. This Connection_Handle must then be used for all the following broadcasts of that type until a reset is performed or the same conflict situation happens again. However, this will occur very rarely.</p> <p>The Host Controller must, in the above conflict case, be able to distinguish between the Broadcast message sent by the Host and the new connection made (this could be even a new SCO link) even though the connection handles are the same.</p> <p>For an HCI Data Packet sent from the Host Controller to the Host where the Broadcast_Flag is 01 or 10, the Connection_Handle parameter should contain the connection handle for the ACL connection to the master that sent the broadcast.</p> <p>Note: Connection handles used for Broadcast do not identify an ACL point-to-point connection, so they must not be used in any command having a Connection_Handle parameter and they will not be returned in any event having a Connection_Handle parameter except the Number Of Completed Packets event.</p>

Flags:**Size: 2 Bits**

The Flag Bits consist of the Packet_Boundary_Flag and Broadcast_Flag. The Packet_Boundary_Flag is located in bit 4 and bit 5, and the Broadcast_Flag is located in bit 6 and 7 in the second byte of the HCI ACL Data packet.

*Packet_Boundary_Flag:**Size: 2 Bits*

Value	Parameter Description
00	Reserved for future use
01	Continuing fragment packet of Higher Layer Message
10	First packet of Higher Layer Message (i.e. start of an L2CAP packet)
11	Reserved for future use

*Broadcast_Flag (in packet from Host to Host Controller):**Size: 2 Bits*

Value	Parameter Description
00	No broadcast. Only point-to-point.
01	Active Broadcast: packet is sent to all active slaves.
10	Piconet Broadcast: packet is sent to all slaves, including slaves in 'Park' mode.
11	Reserved for future use.

*Broadcast_Flag (in packet from Host Controller to Host):**Size: 2 Bits*

Value	Parameter Description
00	Point-to-point
01	Packet received at an active slave (either Active Broadcast or Piconet Broadcast)
10	Packet received at a slave in 'Park' mode (Piconet Broadcast)
11	Reserved for future use.

*Data_Total_Length:**Size: 2 Bytes*

Value	Parameter Description
0xFFFF	Length of data measured in bytes.

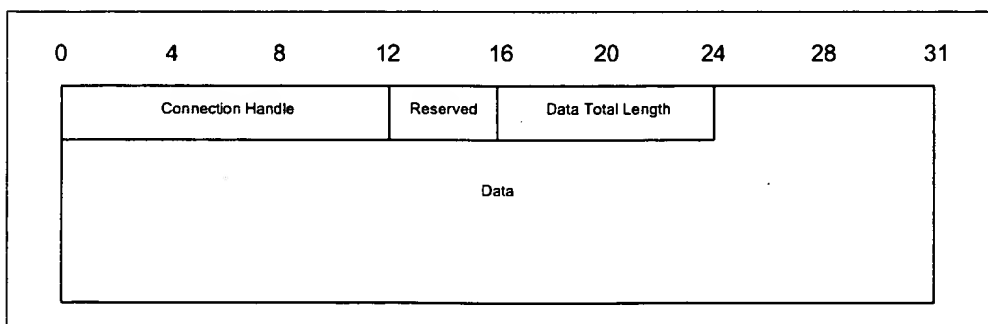


Figure 4.4: HCI SCO Data Packet

*Connection_Handle:**Size: 12 Bits*

Value	Parameter Description
0xXXX	Connection handle to be used to for transmitting a SCO data packet or segment. Range: 0x0000-0x0EFF (0x0F00- 0x0FFF Reserved for future use)

The Reserved Bits consist of four bits which are located from bit 4 to bit 7 in the second byte of the HCI SCO Data packet.

*Reserved:**Size: 4 Bits*

Value	Parameter Description
XXXX	Reserved for future use.

*Data_Total_Length:**Size: 1 Byte*

Value	Parameter Description
0xXX	Length of SCO data measured in bytes

4.5 LINK CONTROL COMMANDS

The Link Control commands allow the Host Controller to control connections to other Bluetooth devices. When the Link Control commands are used, the Link Manager (LM) controls how the Bluetooth piconets and scatternets are established and maintained. These commands instruct the LM to create and modify link layer connections with Bluetooth remote devices, perform Inquiries of other Bluetooth devices in range, and other LMP commands. For the Link Control commands, the OGF is defined as 0x01.

Command	Command Summary Description
Inquiry	The Inquiry command will cause the Bluetooth device to enter Inquiry Mode. Inquiry Mode is used to discovery other nearby Bluetooth devices.
Inquiry_Cancel	The Inquiry_Cancel command will cause the Bluetooth device to stop the current Inquiry if the Bluetooth device is in Inquiry Mode.
Periodic_Inquiry_Mode	The Periodic_Inquiry_Mode command is used to configure the Bluetooth device to perform an automatic Inquiry based on a specified period range.
Exit_Periodic_Inquiry_Mode	The Exit_Periodic_Inquiry_Mode command is used to end the Periodic Inquiry mode when the local device is in Periodic Inquiry Mode.
Create_Connection	The Create_Connection command will cause the link manager to create an ACL connection to the Bluetooth device with the BD_ADDR specified by the command parameters.
Disconnect	The Disconnect command is used to terminate an existing connection.
Add_SCO_Connection	The Add_SCO_Connection command will cause the link manager to create a SCO connection using the ACL connection specified by the Connection Handle command parameter.
Accept_Connection_Request	The Accept_Connection_Request command is used to accept a new incoming connection request.
Reject_Connection_Request	The Reject_Connection_Request command is used to decline a new incoming connection request.
Link_Key_Request_Reply	The Link_Key_Request_Reply command is used to reply to a Link Key Request event from the Host Controller, and specifies the Link Key stored on the Host to be used as the link key for the connection with the other Bluetooth device specified by BD_ADDR.

Command	Command Summary Description
Link_Key_Request_Negative_Reply	The Link_Key_Request_Negative_Reply command is used to reply to a Link Key Request event from the Host Controller if the Host does not have a stored Link Key for the connection with the other Bluetooth Device specified by BD_ADDR.
PIN_Code_Request_Reply	The PIN_Code_Request_Reply command is used to reply to a PIN Code Request event from the Host Controller and specifies the PIN code to use for a connection.
PIN_Code_Request_Negative_Reply	The PIN_Code_Request_Negative_Reply command is used to reply to a PIN Code Request event from the Host Controller when the Host cannot specify a PIN code to use for a connection.
Change_Connection_Packet_Type	The Change_Connection_Packet_Type command is used to change which packet types can be used for a connection that is currently established.
Authentication_Requested	The Authentication_Requested command is used to establish authentication between the two devices associated with the specified Connection Handle.
Set_Connection_Encryption	The Set_Connection_Encryption command is used to enable and disable the link level encryption.
Change_Connection_Link_Key	The Change_Connection_Link_Key command is used to force both devices of a connection associated to the connection handle, to generate a new link key.
Master_Link_Key	The Master_Link_Key command is used to force both devices of a connection associated to the connection handle to use the temporary link key of the Master device or the regular link keys.
Remote_Name_Request	The Remote_Name_Request command is used to obtain the user-friendly name of another Bluetooth device.
Read_Remote_Supported_Features	The Read_Remote_Supported_Features command requests a list of the supported features of a remote device.
Read_Remote_Version_Information	The Read_Remote_Version_Information command will read the values for the version information for the remote Bluetooth device.
Read_Clock_Offset	The Read_Clock_Offset command allows the Host to read the clock offset of remote devices.

4.5.1 Inquiry

Command	OCF	Command Parameters	Return Parameters
HCI_Inquiry	0x0001	LAP, Inquiry_Length, Num_Responses	

Description:

This command will cause the Bluetooth device to enter Inquiry Mode. Inquiry Mode is used to discover other nearby Bluetooth devices. The LAP input parameter contains the LAP from which the inquiry access code shall be derived when the inquiry procedure is made. The Inquiry_Length parameter specifies the total duration of the Inquiry Mode and, when this time expires, Inquiry will be halted. The Num_Responses parameter specifies the number of responses that can be received before the Inquiry is halted. A Command Status event is sent from the Host Controller to the Host when the Inquiry command has been started by the Bluetooth device. When the Inquiry process is completed, the Host Controller will send an Inquiry Complete event to the Host indicating that the Inquiry has finished. The event parameters of Inquiry Complete event will have a summary of the result from the Inquiry process, which reports the number of nearby Bluetooth devices that responded. When a Bluetooth device responds to the Inquiry message, an Inquiry Result event will occur to notify the Host of the discovery.

A device which responds during an inquiry or inquiry period should always be reported to the Host in an Inquiry Result event if the device has not been reported earlier during the current inquiry or inquiry period and the device has not been filtered out using the command Set_Event_Filter. If the device has been reported earlier during the current inquiry or inquiry period, it may or may not be reported depending on the implementation (depending on if earlier results have been saved in the Host Controller and in that case how many responses that have been saved). It is recommended that the Host Controller tries to report a particular device only once during an inquiry or inquiry period.

Command Parameters:**LAP:****Size: 3 Bytes**

Value	Parameter Description
0x9E8B00– 0X9E8B3F	This is the LAP from which the inquiry access code should be derived when the inquiry procedure is made; see "Bluetooth Assigned Numbers" on page 1009.

Inquiry_Length:**Size: 1 Byte**

Value	Parameter Description
N = 0xXX	Maximum amount of time specified before the Inquiry is halted. Size: 1 byte Range: 0x01 – 0x30 Time = N * 1.28 sec Range: 1.28 – 61.44 Sec

Num_Responses:**Size: 1 Byte**

Value	Parameter Description
0x00	Default. Unlimited number of responses.
0xXX	Maximum number of responses from the Inquiry before the Inquiry is halted. Range: 0x01 – 0xFF

Return Parameters:

None.

Event(s) generated (unless masked away):

A Command Status event is sent from the Host Controller to the Host when the Host Controller has started the Inquiry process. An Inquiry Result event will be created for each Bluetooth device which responds to the Inquiry message. In addition, multiple Bluetooth devices which respond to the Inquire message may be combined into the same event. An Inquiry Complete event is generated when the Inquiry process has completed.

Note: no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Inquiry Complete event will indicate that this command has been completed. No Inquiry Complete event will be generated for the canceled Inquiry process.

4.5.2 Inquiry_Cancel

Command	OCF	Command Parameters	Return Parameters
HCI_Inquiry_Cancel	0x0002		Status

Description:

This command will cause the Bluetooth device to stop the current Inquiry if the Bluetooth device is in Inquiry Mode. This command allows the Host to interrupt the Bluetooth device and request the Bluetooth device to perform a different task. The command should only be issued after the Inquiry command has been issued, a Command Status event has been received for the Inquiry command, and before the Inquiry Complete event occurs.

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Inquiry_Cancel command succeeded.
0x01-0xFF	Inquiry_Cancel command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Inquiry Cancel command has completed, a Command Complete event will be generated. No Inquiry Complete event will be generated for the canceled Inquiry process.

4.5.3 Periodic_Inquiry_Mod

Command	OCF	Command Parameters	Return Parameters
HCI_Periodic_Inquiry_Mode	0x0003	Max_Period_Length, Min_Period_Length, LAP, Inquiry_Length, Num_Responses	Status

Description:

The Periodic_Inquiry_Mode command is used to configure the Bluetooth device to enter the Periodic Inquiry Mode that performs an automatic Inquiry. Max_Period_Length and Min_Period_Length define the time range between two consecutive inquiries, from the beginning of an inquiry until the start of the next inquiry. The Host Controller will use this range to determine a new random time between two consecutive inquiries for each Inquiry. The LAP input parameter contains the LAP from which the inquiry access code shall be derived when the inquiry procedure is made. The Inquiry_Length parameter specifies the total duration of the InquiryMode and, when time expires, Inquiry will be halted. The Num_Responses parameter specifies the number of responses that can be received before the Inquiry is halted. This command is completed when the Inquiry process has been started by the Bluetooth device, and a Command Complete event is sent from the Host Controller to the Host. When each of the periodic Inquiry processes are completed, the Host Controller will send an Inquiry Complete event to the Host indicating that the latest periodic Inquiry process has finished. The event parameters of Inquiry Complete event will have a summary of the result from the previous Periodic Inquiry process, which reports the number of nearby Bluetooth devices that responded. When a Bluetooth device responds to the Inquiry message an Inquiry Result event will occur to notify the Host of the discovery.

Note: Max_Period_Length > Min_Period_Length > Inquiry_Length

A device which responds during an inquiry or inquiry period should always be reported to the Host in an Inquiry Result event if the device has not been reported earlier during the current inquiry or inquiry period and the device has not been filtered out using the command Set_Event_Filter. If the device has been reported earlier during the current inquiry or inquiry period, it may or may not be reported depending on the implementation (depending on if earlier results have been saved in the Host Controller and in that case how many responses that have been saved). It is recommended that the Host Controller tries to report a particular device only once during an inquiry or inquiry period.

Command Parameters:

Max_Period_Length:

Size: 2 Bytes

Value	Parameter Description
N = 0xFFFF	Maximum amount of time specified between consecutive inquiries. Size: 2 bytes Range: 0x03 – 0xFFFF Time = N * 1.28 sec Range: 3.84 – 83884.8 Sec 0.0 – 23.3 hours

Min_Period_Length:

Size: 2 Bytes

Value	Parameter Description
N = 0xFFFF	Minimum amount of time specified between consecutive inquiries. Size: 2 bytes Range: 0x02 – 0xFFFE Time = N * 1.28 sec Range: 2.56 – 83883.52 Sec 0.0 – 23.3 hours

LAP:

Size: 3 Bytes

Value	Parameter Description
0x9E8B00– 0x9E8B3F	This is the LAP from which the inquiry access code should be derived when the inquiry procedure is made, see "Bluetooth Assigned Numbers" on page 1009.

Inquiry_Length:

Size: 1 Byte

Value	Parameter Description
N = 0xFF	Maximum amount of time specified before the Inquiry is halted. Size: 1 byte Range: 0x01 – 0xFF Time = N * 1.28 sec Range: 1.28 – 61.44 Sec

Num_Responses:

Size: 1 Byte

Value	Parameter Description
0x00	Default. Unlimited number of responses.
0xFF	Maximum number of responses from the Inquiry before the Inquiry is halted. Range: 0x01 – 0xFF

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Periodic Inquiry Mode command succeeded.
0x01-0xFF	Periodic Inquiry Mode command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

The Periodic Inquiry Mode begins when the Host Controller sends the Command Complete event for this command to the Host. An Inquiry Result event will be created for each Bluetooth device which responds to the Inquiry message. In addition, multiple Bluetooth devices which response to the Inquiry message may be combined into the same event. An Inquiry Complete event is generated when each of the periodic Inquiry processes has completed. No Inquiry Complete event will be generated for the canceled Inquiry process.

4.5.4 Exit_Periodic_Inquiry_Mode

Command	OCF	Command Parameters	Return Parameters
HCI_Exit_Periodic_Inquiry_Mode	0x0004		Status

Description:

The Exit Periodic Inquiry Mode command is used to end the Periodic Inquiry mode when the local device is in Periodic Inquiry Mode. If the local device is currently in an Inquiry process, the Inquiry process will be stopped directly and the Host Controller will no longer perform periodic inquiries until the Periodic Inquiry Mode command is reissued.

Command Parameters:

None.

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Exit Periodic Inquiry Mode command succeeded.
0x01-0xFF	Exit Periodic Inquiry Mode command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

A Command Complete event for this command will occur when the local device is no longer in Periodic Inquiry Mode. No Inquiry Complete event will be generated for the canceled Inquiry process.

4.5.5 Create_Connection

Command	OCF	Command Parameters	Return Parameters
HCI_Create_Connection	0x0005	BD_ADDR, Packet_Type, Page_Scan_Repetition_Mode, Page_Scan_Mode, Clock_Offset, Allow_Role_Switch	

Description:

This command will cause the Link Manager to create a connection to the Bluetooth device with the BD_ADDR specified by the command parameters. This command causes the local Bluetooth device to begin the Page process to create a link level connection. The Link Manager will determine how the new ACL connection is established. This ACL connection is determined by the current state of the device, its piconet, and the state of the device to be connected. The Packet_Type command parameter specifies which packet types the Link Manager shall use for the ACL connection. The Link Manager must use only the packet type(s) specified by the Packet_Type command parameter for sending HCI ACL Data Packets. Multiple packet types may be specified for the Packet_Type parameter by performing a bit-wise OR operation of the different packet types. The Link Manager may choose which packet type to be used from the list of acceptable packet types. The Page_Scan_Repetition_Mode and Page_Scan_Mode parameters specify the page scan modes supported by the remote device with the BD_ADDR. This is the information that was acquired during the inquiry process. The Clock_Offset parameter is the difference between its own clock and the clock of the remote device with BD_ADDR. Only bits 2 through 16 of the difference are used, and they are mapped to this parameter as bits 0 through 14 respectively. A Clock_Offset_Valid_Flag, located in bit 15 of the Clock_Offset parameter, is used to indicate if the Clock_Offset is valid or not. A Connection handle for this connection is returned in the Connection Complete event (see below). The Allow_Role_Switch parameter specifies if the local device accepts or rejects the request of a master-slave role switch when the remote device requests it at the connection setup (in the Role parameter of the Accept_Connection_Request command) (before the local Host Controller returns a Connection Complete event). For a definition of the different packet types see the "Baseband Specification" on page 33.

Note: At least one packet type must be specified. The Host should enable as many packet types as possible for the Link Manager to perform efficiently. However, the Host must not enable packet types that the local device does not support.

Command Parameters:**BD_ADDR:****Size: 6 Bytes**

Value	Parameter Description
0XXXXXXXXXXXXX	BD_ADDR of the Device to be connected.

Packet_Type:**Size: 2 Bytes**

Value	Parameter Description
0x0001	Reserved for future use.
0x0002	Reserved for future use.
0x0004	Reserved for future use.
0x0008	DM1
0x0010	DH1
0x0020	Reserved for future use.
0x0040	Reserved for future use.
0x0080	Reserved for future use.
0x0100	Reserved for future use.
0x0200	Reserved for future use.
0x0400	DM3
0x0800	DH3
0x1000	Reserved for future use.
0x2000	Reserved for future use.
0x4000	DM5
0x8000	DH5

Page_Scan_Repetition_Mode:**Size: 1 Byte**

Value	Parameter Description
0x00	R0
0x01	R1
0x02	R2
0x03 – 0xFF	Reserved.

Page_Scan_Mode:**Size: 1 Byte**

Value	Parameter Description
0x00	Mandatory Page Scan Mode.
0x01	Optional Page Scan Mode I.
0x02	Optional Page Scan Mode II.
0x03	Optional Page Scan Mode III.
0x04 – 0xFF	Reserved.

Clock_Offset:**Size: 2 Bytes**

Bit format	Parameter Description
Bit 14.0	Bit 16.2 of CLKslave-CLKmaster.
Bit 15	Clock_Offset_Valid_Flag Invalid Clock Offset = 0 Valid Clock Offset = 1

Allow_Role_Switch:**Size: 1 Byte**

Value	Parameter Description
0x00	The local device will be a master, and will not accept a master-slave switch requested by the remote device at the connection setup.
0x01	The local device may be a master, or may become a slave after accepting a master-slave switch requested by the remote device at the connection setup.
0x02-0xFF	Reserved for future use.

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Create Connection command, the Host Controller sends the Command Status event to the Host. In addition, when the LM determines the connection is established, the Host Controller, on both Bluetooth devices that form the connection, will send a Connection Complete event to each Host. The Connection Complete event contains the Connection Handle if this command is successful.

Note: no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Connection Complete event will indicate that this command has been completed.

4.5.6 Disconnect

Command	OCF	Command Parameters	Return Parameters
HCI_Disconnect	0x0006	Connection_Handle, Reason	

Description:

The Disconnection command is used to terminate an existing connection. The Connection_Handle command parameter indicates which connection is to be disconnected. The Reason command parameter indicates the reason for ending the connection. The remote Bluetooth device will receive the Reason command parameter in the Disconnection Complete event. All SCO connections on a physical link should be disconnected before the ACL connection on the same physical connection is disconnected.

Command Parameters:*Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xXXXX	Connection Handle for the connection being disconnected. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

*Reason:**Size: 1 Byte*

Value	Parameter Description
0x13-0x15, 0x1A	Other End Terminated Connection error codes (0x13-0x15) and Unsupported Remote Feature error code (0x1A) see Table 6.1 on page 745 for list of Error Codes.

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Disconnect command, it sends the Command Status event to the Host. The Disconnection Complete event will occur at each Host when the termination of the connection has completed, and indicates that this command has been completed.

Note: No Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Disconnection Complete event will indicate that this command has been completed.

4.5.7 Add_SCO_Connection

Command	OCF	Command Parameters	Return Parameters
HCI_Add_SCO_Connection	0x0007	Connection_Handle, Packet_Type	

Description:

This command will cause the link manager to create a SCO connection using the ACL connection specified by the Connection_Handle command parameter. This command causes the local Bluetooth device to create a SCO connection. The Link Manager will determine how the new connection is established. This connection is determined by the current state of the device, its piconet, and the state of the device to be connected. The Packet_Type command parameter specifies which packet types the Link Manager should use for the connection. The Link Manager must only use the packet type(s) specified by the Packet_Type command parameter for sending HCI SCO Data Packets. Multiple packet types may be specified for the Packet_Type command parameter by performing a bitwise OR operation of the different packet types. The Link Manager may choose which packet type is to be used from the list of acceptable packet types. A Connection Handle for this connection is returned in the Connection Complete event (see below).

Note: SCO-Connection can only be created when an ACL Connection already exists. For a definition of the different packet types, see the "Baseband Specification" on page 33.

Note: At least one packet type must be specified. The Host should enable as many packet types as possible for the Link Manager to perform efficiently. However, the Host must not enable packet types that the local device does not support.

Command Parameters:*Connection_Handle**Size 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xFFFF	Connection Handle for the ACL connection being used to create an SCO connection. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

*Packet_Type:**Size: 2 Bytes*

Value	Parameter Description
0x0001	Reserved for future use.
0x0002	Reserved for future use.
0x0004	Reserved for future use.
0x0008	Reserved for future use.
0x0010	Reserved for future use.
0x0020	HV1
0x0040	HV2
0x0080	HV3
0x0100	Reserved for future use.
0x0200	Reserved for future use.
0x0400	Reserved for future use.
0x0800	Reserved for future use.
0x1000	Reserved for future use.
0x2000	Reserved for future use.
0x4000	Reserved for future use.
0x8000	Reserved for future use.

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Add_SCO_Connection command, it sends the Command Status event to the Host. In addition, when the LM determines the connection is established, the Host Controller, on both Bluetooth devices that form the connection, will send a Connection Complete event to each Host. The Connection Complete event contains the Connection Handle if this command is successful.

Note: no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Connection Complete event will indicate that this command has been completed.

4.5.8 Accept_Connection_Request

Command	OCF	Command Parameters	Return Parameters
HCI_Accept_Connection_Request	0x0009	BD_ADDR, Role	

Description:

The Accept_Connection_Request command is used to accept a new incoming connection request. The Accept_Connection_Request command shall only be issued after a Connection Request event has occurred. The Connection Request event will return the BD_ADDR of the device which is requesting the connection. This command will cause the Link Manager to create a connection to the Bluetooth device, with the BD_ADDR specified by the command parameters. The Link Manager will determine how the new connection will be established. This will be determined by the current state of the device, its piconet, and the state of the device to be connected. The Role command parameter allows the Host to specify if the Link Manager shall perform a Master-Slave switch, and become the Master for this connection. Also, the decision to accept a connection must be completed before the connection accept timeout expires on the local Bluetooth Module.

Note: when accepting SCO connection request, the Role parameter is not used and will be ignored by the Host Controller.

Command Parameters:**BD_ADDR:***Size: 6 Bytes*

Value	Parameter Description
0xxxxxxxxxxxxx	BD_ADDR of the Device to be connected

Role:*Size: 1 Byte*

Value	Parameter Description
0x00	Become the Master for this connection. The LM will perform the Master/Slave switch.
0x01	Remain the Slave for this connection. The LM will NOT perform the Master/Slave switch.

Return Parameters:

None.

Event(s) generated (unless masked away):

The `Accept_Connection_Request` command will cause the `Command Status` event to be sent from the Host Controller when the Host Controller begins setting up the connection. In addition, when the Link Manager determines the connection is established, the Host Controllers on both Bluetooth devices that form the connection will send a `Connection Complete` event to each Host. The `Connection Complete` event contains the `Connection Handle` if this command is successful.

Note: no `Command Complete` event will be sent by the Host Controller to indicate that this command has been completed. Instead, the `Connection Complete` event will indicate that this command has been completed.

4.5.9 Reject_Connection_Request

Command	OCF	Command Parameters	Return Parameters
HCI_Reject_Connection_Request	0x000A	BD_ADDR, Reason	

Description:

The Reject_Connection_Request command is used to decline a new incoming connection request. The Reject_Connection_Request command shall only be called after a Connection Request event has occurred. The Connection Request event will return the BD_ADDR of the device that is requesting the connection. The Reason command parameter will be returned to the connecting device in the Status parameter of the Connection Complete event returned to the Host of the connection device, to indicate why the connection was declined.

Command Parameters:**BD_ADDR:***Size: 6 Bytes*

Value	Parameter Description
0XXXXXXXXXXXXX	BD_ADDR of the Device to reject the connection from.

Reason:*Size: 1 Byte*

Value	Parameter Description
0x0D-0x0F	Host Reject Error Code. See Table 6.1 on page 745 for list of Error Codes and descriptions.

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Reject_Connection_Request command, the Host Controller sends the Command Status event to the Host. A Connection Complete event will then be sent, both to the local Host and to the Host of the device attempting to make the connection. The Status parameter of the Connection Complete event, which is sent to the Host of the device attempting to make the connection, will contain the Reason command parameter from this command.

Note: no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Connection Complete event will indicate that this command has been completed.

4.5.10 Link_Key_Request_Reply

Command	OCF	Command Parameters	Return Parameters
HCI_Link_Key_Request_Reply	0x000B	BD_ADDR, Link_Key	Status, BD_ADDR

Description:

The Link_Key_Request_Reply command is used to reply to a Link Key Request event from the Host Controller, and specifies the Link Key stored on the Host to be used as the link key for the connection with the other Bluetooth Device specified by BD_ADDR. The Link Key Request event will be generated when the Host Controller needs a Link Key for a connection.

When the Host Controller generates a Link Key Request event in order for the local Link Manager to respond to the request from the remote Link Manager (as a result of a Create_Connection or Authentication_Requested command from the remote Host), the local Host must respond with either a Link_Key_Request_Reply or Link_Key_Request_Negative_Reply command before the remote Link Manager detects LMP response timeout. (See "Link Manager Protocol" on page 185.)

Command Parameters:**BD_ADDR:***Size: 6 Bytes*

Value	Parameter Description
0xxxxxxxxxxxxx	BD_ADDR of the Device of which the Link Key is for.

Link_Key:*Size: 16 Bytes*

Value	Parameter Description
0xxxxxxxxxxxxx xxxxxxxxxxxxx xxxxxxxxxxxxx	Link Key for the associated BD_ADDR.

Return Parameters:**Status:***Size: 1 Byte*

Value	Parameter Description
0x00	Link_Key_Request_Reply command succeeded.
0x01-0xFF	Link_Key_Request_Reply command failed. See Table 6.1 on page 745 for list of Error Codes.

BD_ADDR:

Size: 6 Bytes

Value	Parameter Description
0XXXXXXXXX XXXX	BD_ADDR of the Device of which the Link Key request reply has completed.

Event(s) generated (unless masked away):

The Link_Key_Request_Reply command will cause a Command Complete event to be generated.

4.5.11 Link_Key_Request_Negative_Reply

Command	OCF	Command Parameters	Return Parameters
HCI_Link_Key_Request_Negative_Reply	0x000C	BD_ADDR	Status, BD_ADDR

Description:

The Link_Key_Request_Negative_Reply command is used to reply to a Link Key Request event from the Host Controller if the Host does not have a stored Link Key for the connection with the other Bluetooth Device specified by BD_ADDR. The Link Key Request event will be generated when the Host Controller needs a Link Key for a connection.

When the Host Controller generates a Link Key Request event in order for the local Link Manager to respond to the request from the remote Link Manager (as a result of a Create_Connection or Authentication_Requested command from the remote Host), the local Host must respond with either a Link_Key_Request_Reply or Link_Key_Request_Negative_Reply command before the remote Link Manager detects LMP response timeout. (See "Link Manager Protocol" on page 185.)

Command Parameters:**BD_ADDR:***Size: 6 Bytes*

Value	Parameter Description
0XXXXXXXXX XX	BD_ADDR of the Device which the Link Key is for.

Return Parameters:**Status:***Size: 1 Byte*

Value	Parameter Description
0x00	Link_Key_Request_Negative_Reply command succeeded.
0x01-0xFF	Link_Key_Request_Negative_Reply command failed. See Table 6.1 on page 745 for list of Error Codes.

BD_ADDR:*Size: 6 Bytes*

Value	Parameter Description
0XXXXXXXXX XXXX	BD_ADDR of the Device which the Link Key request negative reply has completed.

Event(s) generated (unless masked away):

The Link_Key_Request_Negative_Reply command will cause a Command Complete event to be generated.

4.5.12 PIN_Code_Request_Reply

Command	OCF	Command Parameters	Return Parameters
HCI_PIN_Code_Request_Reply	0x000D	BD_ADDR, PIN_Code_Length, PIN_Code	Status, BD_ADDR

Description:

The PIN_Code_Request_Reply command is used to reply to a PIN Code request event from the Host Controller, and specifies the PIN code to use for a connection. The PIN Code Request event will be generated when a connection with remote initiating device has requested pairing.

When the Host Controller generates a PIN Code Request event in order for the local Link Manager to respond to the request from the remote Link Manager (as a result of a Create_Connection or Authentication_Requested command from the remote Host), the local Host must respond with either a PIN_Code_Request_Reply or PIN_Code_Request_Negative_Reply command before the remote Link Manager detects LMP response timeout. (See "Link Manager Protocol" on page 185.)

Command Parameters:**BD_ADDR:****Size: 6 Bytes**

Value	Parameter Description
0XXXXXXXXXX XX	BD_ADDR of the Device which the PIN code is for.

PIN_Code_Length:**Size: 1 Byte**

Value	Parameter Description
0xXX	The PIN code length specifies the length, in bytes, of the PIN code to be used. Range: 0x01-0x10

PIN_Code:**Size: 16 Bytes**

Value	Parameter Description
0XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX	PIN code for the device that is to be connected. The Host should insure that strong PIN Codes are used. PIN Codes can be up to a maximum of 128 bits. The MSB of the PIN Code occupies byte zero.

Return Parameters:**Status:****Size: 1 Byte**

Value	Parameter Description
0x00	PIN_Code_Request_Reply command succeeded.
0x01-0xFF	PIN_Code_Request_Reply command failed. See Table 6.1 on page 745 for list of Error Codes.

BD_ADDR:**Size: 6 Bytes**

Value	Parameter Description
0xFFFFFFFF XXXX	BD_ADDR of the Device which the PIN Code request reply has completed.

Event(s) generated (unless masked away):

The PIN_Code_Request_Reply command will cause a Command Complete event to be generated.

4.5.13 PIN_Code_Request_N gative_Reply

Command	OCF	Command Parameters	Return Parameters
HCI_PIN_Code_Request_Negative_Reply	0x000E	BD_ADDR	Status, BD_ADDR

Description:

The PIN_Code_Request_Negative_Reply command is used to reply to a PIN Code request event from the Host Controller when the Host cannot specify a PIN code to use for a connection. This command will cause the pair request with remote device to fail.

When the Host Controller generates a PIN Code Request event in order for the local Link Manager to respond to the request from the remote Link Manager (as a result of a Create_Connection or Authentication_Requested command from the remote Host), the local Host must respond with either a PIN_Code_Request_Reply or PIN_Code_Request_Negative_Reply command before the remote Link Manager detects LMP response timeout. (See "Link Manager Protocol" on page 185.)

Command Parameters:**BD_ADDR:***Size: 6 Bytes*

Value	Parameter Description
0XXXXXXXXXXXXX	BD_ADDR of the Device which this command is responding to.

Return Parameters:**Status:***Size: 1 Byte*

Value	Parameter Description
0x00	PIN_Code_Request_Negative_Reply command succeeded.
0x01-0xFF	PIN_Code_Request_Negative_Reply command failed. See Table 6.1 on page 7450 for list of Error Codes.

BD_ADDR:*Size: 6 Bytes*

Value	Parameter Description
0XXXXXXXXX XXXX	BD_ADDR of the Device which the PIN Code request negative reply has completed.

Event(s) generated (unless masked away):

The PIN_Code_Request_Negative_Reply command will cause a Command Complete event to be generated.

4.5.14 Change_Connection_Packet_Type

Command	OCF	Command Parameters	Return Parameters
HCI_Change_Connection_Packet_Type	0x000F	Connection_Handle, Packet_Type	

Description:

The Change_Connection_Packet_Type command is used to change which packet types can be used for a connection that is currently established. This allows current connections to be dynamically modified to support different types of user data. The Packet_Type command parameter specifies which packet types the Link Manager can use for the connection. The Link Manager must only use the packet type(s) specified by the Packet_Type command parameter for sending HCI Data Packets. The interpretation of the value for the Packet_Type command parameter will depend on the Link_Type command parameter returned in the Connection Complete event at the connection setup. Multiple packet types may be specified for the Packet_Type command parameter by bitwise OR operation of the different packet types. For a definition of the different packet types see the "Baseband Specification" on page 33.

Note: At least one packet type must be specified. The Host should enable as many packet types as possible for the Link Manager to perform efficiently. However, the Host must not enable packet types that the local device does not support.

Command Parameters:*Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xXXXX	Connection Handle to be used to for transmitting and receiving voice or data. Returned from creating a connection. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

*Packet_Type:**Size: 2 Bytes**For ACL Link_Type*

Value	Parameter Description
0x0001	Reserved for future use.
0x0002	Reserved for future use.
0x0004	Reserved for future use.
0x0008	DM1

Value	Parameter Description
0x0010	DH1
0x0020	Reserved for future use.
0x0040	Reserved for future use.
0x0080	Reserved for future use.
0x0100	Reserved for future use.
0x0200	Reserved for future use.
0x0400	DM3
0x0800	DH3
0x1000	Reserved for future use.
0x2000	Reserved for future use.
0x4000	DM5
0x8000	DH5

For SCO Link Type

Value	Parameter Description
0x0001	Reserved for future use.
0x0002	Reserved for future use.
0x0004	Reserved for future use.
0x0008	Reserved for future use.
0x0010	Reserved for future use.
0x0020	HV1
0x0040	HV2
0x0080	HV3
0x0100	Reserved for future use.
0x0200	Reserved for future use.
0x0400	Reserved for future use.
0x0800	Reserved for future use.
0x1000	Reserved for future use.
0x2000	Reserved for future use.
0x4000	Reserved for future use.
0x8000	Reserved for future use.

R turn Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Change Connection Packet Type command, the Host Controller sends the Command Status event to the Host. In addition, when the Link Manager determines the packet type has been changed for the connection, the Host Controller on the local device will send a Connection Packet Type Changed event to the Host. This will be done at the local side only.

Note: no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Connection Packet Type Changed event will indicate that this command has been completed.

4.5.15 Authentication_Requested

Command	OCF	Command Parameters	Return Parameters
HCI_Authentication_Requested	0x0011	Connection_Handle	

Description:

The Authentication_Requested command is used to try to authenticate the remote device associated with the specified Connection Handle. The Host must not issue the Authentication_Requested command with a Connection_Handle corresponding to an encrypted link. On an authentication failure, the Host Controller or Link Manager shall not automatically detach the link. The Host is responsible for issuing a Disconnect command to terminate the link if the action is appropriate.

Note: the Connection_Handle command parameter is used to identify the other Bluetooth device, which forms the connection. The Connection Handle should be a Connection Handle for an ACL connection.

Command Parameters:

Connection_Handle:

Size 2 Bytes (12 Bits meaningful)

Value	Parameter Description
0xXXXX	Connection Handle to be used to set up authentication for all Connection Handles with the same Bluetooth device end-point as the specified Connection Handle. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Authentication_Requested command, it sends the Command Status event to the Host. The Authentication Complete event will occur when the authentication has been completed for the connection and is indication that this command has been completed.

Note: no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Authentication Complete event will indicate that this command has been completed.

Note: When the local or remote Host Controller does not have a link key for the specified Connection_Handle, it will request the link key from its Host, before the local Host finally receives the Authentication Complete event.

4.5.16 Set_Connection_Encryption

Command	OCF	Command Parameters	Return Parameters
HCI_Set_Connection_Encryption	0x0013	Connection_Handle, Encryption_Enable	

Description:

The Set_Connection_Encryption command is used to enable and disable the link level encryption. Note: the Connection_Handle command parameter is used to identify the other Bluetooth device which forms the connection. The Connection Handle should be a Connection Handle for an ACL connection. While the encryption is being changed, all ACL traffic must be turned off for all Connection Handles associated with the remote device.

Command Parameters:

Connection_Handle: *Size 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xXXXX	Connection Handle to be used to enable/disable the link layer encryption for all Connection Handles with the same Bluetooth device end-point as the specified Connection Handle. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Encryption_Enable: *Size: 1 Byte*

Value	Parameter Description
0x00	Turn Link Level Encryption OFF.
0x01	Turn Link Level Encryption ON.

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Set_Connection_Encryption command, the Host Controller sends the Command Status event to the Host. When the Link Manager has completed enabling/disabling encryption for the connection, the Host Controller on the local Bluetooth device will send an Encryption Change event to the Host, and the Host Controller on the remote device will also generate an Encryption Change event.

Note: no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Encryption Change event will indicate that this command has been completed.

4.5.17 Change_Connection_Link_Key

Command	OCF	Command Parameters	Return Parameters
HCI_Change_Connection_Link_Key	0x0015	Connection_Handle	

Description:

The Change_Connection_Link_Key command is used to force both devices of a connection associated with the connection handle to generate a new link key. The link key is used for authentication and encryption of connections.

Note: the Connection_Handle command parameter is used to identify the other Bluetooth device forming the connection. The Connection Handle should be a Connection Handle for an ACL connection. If the connection encryption is enabled, and the temporary link key is currently used, then the Bluetooth master device will automatically restart the encryption.

Command Parameters:

Connection_Handle:

Size 2 Bytes (12 Bits meaningful)

Value	Parameter Description
0xXXXX	Connection Handle used to identify a connection. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Change_Connection_Link_Key command, the Host Controller sends the Command Status event to the Host. When the Link Manager has changed the Link Key for the connection, the Host Controller on the local Bluetooth device will send a Link Key Notification event and a Change Connection Link Key Complete event to the Host, and the Host Controller on the remote device will also generate a Link Key Notification event. The Link Key Notification event indicates that a new connection link key is valid for the connection.

Note: no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Change Connection Link Key Complete event will indicate that this command has been completed.

4.5.18 Master_Link_Key

Command	OCF	Command Parameters	Return Parameters
HCI_Master_Link_Key	0x0017	Key_Flag	

Description:

The Master Link Key command is used to force the device that is master of the piconet to use the temporary link key of the master device, or the semi-permanent link keys. The temporary link key is used for encryption of broadcast messages within a piconet, and the semi-permanent link keys are used for private encrypted point-to-point communication. The Key_Flag command parameter is used to indicate which Link Key (temporary link key of the Master, or the semi-permanent link keys) shall be used.

Command Parameters:*Key_Flag:**Size: 1 Byte*

Value	Parameter Description
0x00	Use semi-permanent Link Keys.
0x01	Use Temporary Link Key.

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Master_Link_Key command, the Host Controller sends the Command Status event to the Host. When the Link Manager has changed link key, the Host Controller on both the local and the remote device will send a Master Link Key Complete event to the Host. The Connection Handle on the master side will be a Connection Handle for one of the existing connections to a slave. On the slave side, the Connection Handle will be a Connection Handle to the initiating master.

The Master Link Key Complete event contains the status of this command.

Note: no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Master Link Key Complete event will indicate that this command has been completed.

4.5.19 R mot _Nam _R qu st

Command	OCF	Command Parameters	Return Parameters
HCI_Remote_Name_Request	0x0019	BD_ADDR, Page_Scan_Repetition_Mode, Page_Scan_Mode, Clock_Offset	

Description:

The Remote_Name_Request command is used to obtain the user-friendly name of another Bluetooth device. The user-friendly name is used to enable the user to distinguish one Bluetooth device from another. The BD_ADDR command parameter is used to identify the device for which the user-friendly name is to be obtained. The Page_Scan_Repetition_Mode and Page_Scan_Mode command parameters specify the page scan modes supported by the remote device with the BD_ADDR. This is the information that was acquired during the inquiry process. The Clock_Offset parameter is the difference between its own clock and the clock of the remote device with BD_ADDR. Only bits 2 through 16 of the difference are used and they are mapped to this parameter as bits 0 through 14 respectively. A Clock_Offset_Valid_Flag, located in bit 15 of the Clock_Offset command parameter, is used to indicate if the Clock Offset is valid or not.

Note: if no connection exists between the local device and the device corresponding to the BD_ADDR, a temporary link layer connection will be established to obtain the name of the remote device.

Command Parameters:**BD_ADDR:***Size: 6 Bytes*

Value	Parameter Description
0XXXXXXXXXX XX	BD_ADDR for the device whose name is requested.

Page_Scan_Repetition_Mode:*Size: 1 Byte*

Value	Parameter Description
0x00	R0
0x01	R1
0x02	R2
0x03 – 0xFF	Reserved.

Page_Scan_Mode:**Size: 1 Byte**

Value	Parameter Description
0x00	Mandatory Page Scan Mode.
0x01	Optional Page Scan Mode I.
0x02	Optional Page Scan Mode II.
0x03	Optional Page Scan Mode III.
0x04 – 0xFF	Reserved.

Clock_Offset:**Size: 2 Bytes**

Bit format	Parameter Description
Bit 14.0	Bit 16.2 of CLKslave-CLKmaster.
Bit 15	Clock_Offset_Valid_Flag Invalid Clock Offset = 0 Valid Clock Offset = 1

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Remote_Name_Request command, the Host Controller sends the Command Status event to the Host. When the Link Manager has completed the LMP messages to obtain the remote name, the Host Controller on the local Bluetooth device will send a Remote Name Request Complete event to the Host. Note: no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, only the Remote Name Request Complete event will indicate that this command has been completed.

4.5.20 Read_R mote_Supported_Featur s

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Remote_Supported_Features	0x001B	Connection_Handle	

Description:

This command requests a list of the supported features for the remote device identified by the Connection_Handle parameter. The Connection_Handle must be a Connection_Handle for an ACL connection. The Read Remote Supported Features Complete event will return a list of the LMP features. For details see "Link Manager Protocol" on page 185.

Command Parameters:*Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xXXXX	Specifies which Connection_Handle's LMP-supported features list to get. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Read_Remote_Supported_Features command, the Host Controller sends the Command Status event to the Host. When the Link Manager has completed the LMP messages to determine the remote features, the Host Controller on the local Bluetooth device will send a Read Remote Supported Features Complete event to the Host. The Read Remote Supported Features Complete event contains the status of this command, and parameters describing the supported features of the remote device. Note: no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Read Remote Supported Features Complete event will indicate that this command has been completed.

4.5.21 Read_Remote_Version_Information

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Remote_Version_Information	0x001D	Connection_Handle	

Description:

This command will obtain the values for the version information for the remote Bluetooth device identified by the Connection_Handle parameter. The Connection_Handle must be a Connection_Handle for an ACL connection.

Command Parameters:

Connection_Handle:

Size: 2 Bytes (12 Bits meaningful)

Value	Parameter Description
0xXXXX	Specifies which Connection Handle's version information to get. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Read_Remote_Version_Information command, the Host Controller sends the Command Status event to the Host. When the Link Manager has completed the LMP messages to determine the remote version information, the Host Controller on the local Bluetooth device will send a Read Remote Version Information Complete event to the Host. The Read Remote Version Information Complete event contains the status of this command, and parameters describing the version and subversion of the LMP used by the remote device.

Note: no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the Read Remote Version Information Complete event will indicate that this command has been completed.

4.5.22 Read_Clock_Offset

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Clock_Offset	0x001F	Connection_Handle	

Description:

Both the System Clock and the clock offset of a remote device are used to determine what hopping frequency is used by a remote device for page scan. This command allows the Host to read clock offset of remote devices. The Connection_Handle must be a Connection_Handle for an ACL connection. This command could be used to facilitate handoffs of Bluetooth devices from one device to another.

Command Parameters:*Connection_Handle:**Size: 2 Bytes (12 bits meaningful)*

Value	Parameter Description
0xXXXX	Specifies which Connection Handle's Clock Offset parameter is returned. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the Read_Clock_Offset command, the Host Controller sends the Command Status event to the Host. If this command was requested at the master and the Link Manager has completed the LMP messages to obtain the Clock Offset information, the Host Controller on the local Bluetooth device will send a Read Clock Offset Complete event to the Host. Note: no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, only the Read Clock Offset Complete event will indicate that this command has been completed. If the command is requested at the slave, the LM will immediately send a Command Status event and a Read Clock Offset Complete event to the Host, without an exchange of LMP PDU.

4.6 LINK POLICY COMMANDS

The Link Policy Commands provide methods for the Host to affect how the Link Manager manages the piconet. When Link Policy Commands are used, the LM still controls how Bluetooth piconets and scatternets are established and maintained, depending on adjustable policy parameters. These policy commands modify the Link Manager behavior that can result in changes to the link layer connections with Bluetooth remote devices.

Note: only one ACL connection can exist between two Bluetooth Devices, and therefore there can only be one ACL HCI Connection Handle for each physical link layer Connection. The Bluetooth Host Controller provides policy adjustment mechanisms to provide support for a number of different policies. This capability allows one Bluetooth module to be used to support many different usage models, and the same Bluetooth module can be incorporated in many different types of Bluetooth devices. For the Link Policy Commands, the OGF is defined as 0x02.

Command	Command Summary Description
Hold_Mode	The Hold_Mode command is used to alter the behavior of the LM and have the LM place the local or remote device into the hold mode.
Sniff_Mode	The Sniff_Mode command is used to alter the behavior of the LM and have the LM place the local or remote device into the sniff mode.
Exit_Sniff_Mode	The Exit_Sniff_Mode command is used to end the sniff mode for a connection handle which is currently in sniff mode.
Park_Mode	The Park_Mode command is used to alter the behavior of the LM and have the LM place the local or remote device into the Park mode.
Exit_Park_Mode	The Exit_Park_Mode command is used to switch the Bluetooth device from park mode back to active mode.
QoS_Setup	The QoS_Setup command is used to specify Quality of Service parameters for a connection handle.
Role_Discovery	The Role_Discovery command is used for a Bluetooth device to determine which role the device is performing for a particular Connection Handle.
Switch_Role	The Switch_Role command is used for a Bluetooth device switch the current role the device is performing for a particular connection with the specified Bluetooth device

Command	Command Summary Description
Read_Link_Policy_Settings	The Read_Link_Policy_Settings command will read the Link Policy settings for the specified Connection Handle. The Link Policy settings allow the Host to specify which Link Modes the LM can use for the specified Connection Handle.
Write_Link_Policy_Settings	The Write_Link_Policy_Settings command will write the Link Policy settings for the specified Connection Handle. The Link Policy settings allow the Host to specify which Link Modes the LM can use for the specified Connection Handle.

4.6.1 Hold_Mode

Command	OCF	Command Parameters	Return Parameters
HCI_Hold_Mode	0x0001	Connection_Handle, Hold_Mode_Max_Interval, Hold_Mode_Min_Interval	

Description:

The Hold_Mode command is used to alter the behavior of the Link Manager, and have it place the ACL baseband connection associated by the specified Connection Handle into the hold mode. The Hold_Mode_Max_Interval and Hold_Mode_Min_Interval command parameters specify the length of time the Host wants to put the connection into the hold mode. The local and remote devices will negotiate the length in the hold mode. The Hold_Mode_Max_Interval parameter is used to specify the maximum length of the Hold interval for which the Host may actually enter into the hold mode after negotiation with the remote device. The Hold interval defines the amount of time between when the Hold Mode begins and when the Hold Mode is completed. The Hold_Mode_Min_Interval parameter is used to specify the minimum length of the Hold interval for which the Host may actually enter into the hold mode after the negotiation with the remote device. Therefore the Hold_Mode_Min_Interval cannot be greater than the Hold_Mode_Max_Interval. The Host Controller will return the actual Hold interval in the Interval parameter of the Mode Change event, if the command is successful. This command enables the Host to support a low-power policy for itself or several other Bluetooth devices, and allows the devices to enter Inquiry Scan, Page Scan, and a number of other possible actions.

Note: the connection handle cannot be of the SCO link type.

Command Parameters:

Connection_Handle:

Size: 2 Bytes (12 Bits meaningful)

Value	Parameter Description
0xXXXX	Connection Handle to be used to identify a connection. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Hold_Mode_Max_Interval:

Size: 2 Bytes

Value	Parameter Description
N = 0xXXXX	Maximum acceptable number of Baseband slots to wait in Hold Mode. Time Length of the Hold = $N * 0.625 \text{ msec}$ (1 Baseband slot) Range for N: 0x0001-0xFFFF Tim Rang : 0.625ms - 40.9 sec

Hold_Mode_Min_Interval:**Size: 2 Bytes**

Value	Parameter Description
N = 0xXXXX	Minimum acceptable number of Baseband slots to wait in Hold Mod . Time Length of the Hold = $N * 0.625$ msec (1 Baseband slot) Range for N: 0x0001-0xFFFF Time Range: 0.625 msec - 40.9 sec

Return Parameters:

None.

Event(s) generated (unless masked away):

The Host Controller sends the Command Status event for this command to the Host when it has received the Hold_Mode command. The Mode Change event will occur when the Hold Mode has started and the Mode Change event will occur again when the Hold Mode has completed for the specified connection handle. The Mode Change event signaling the end of the Hold Mode is an estimation of the hold mode ending if the event is for a remote Bluetooth device. Note: no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, only the Mode Change event will indicate that this command has been completed. If an error occurs after the Command Status event has occurred, then the status in the Mode Change event will indicate the error.

4.6.2 Sniff_Mod

Command	OCF	Command Parameters	Return Parameters
HCI_Sniff_Mode	0x0003	Connection_Handle, Sniff_Max_Interval, Sniff_Min_Interval, Sniff_Attempt, Sniff_Timeout	

Description:

The Sniff Mode command is used to alter the behavior of the Link Manager and have it place the ACL baseband connection associated with the specified Connection Handle into the sniff mode. The Connection_Handle command parameter is used to identify which ACL link connection is to be placed in sniff mode. The Sniff_Max_Interval and Sniff_Min_Interval command parameters are used to specify the requested acceptable maximum and minimum periods in the Sniff Mode. The Sniff_Min_Interval cannot be greater than the Sniff_Max_Interval. The sniff interval defines the amount of time between each consecutive sniff period. The Host Controller will return the actual sniff interval in the Interval parameter of the Mode Change event, if the command is successful. The slave will listen at the end of every actual sniff interval, for the period specified by the Sniff_Attempt command parameter. The slave will continue listening for packets for an additional period specified by Sniff_Timeout, as long as it is receiving packets. This command enables the Host to support a low-power policy for itself or several other Bluetooth devices, and allows the devices to enter Inquiry Scan, Page Scan, and a number of other possible actions.

Note: in addition, the connection handle cannot be one of SCO link type.

Command Parameters:

Connection_Handle:

Size: 2 Bytes (12 Bits meaningful)

Value	Parameter Description
0xXXXX	Connection Handle to be used to identify a connection. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Sniff_Max_Interval:

Size: 2 Byte

Value	Parameter Description
N = 0xXXXX	Maximum acceptable number of Baseband slots between each sniff period. (Sniff_Max_Interval >= Sniff_Min_Interval) Length = N * 0.625 msec (1 Baseband slot) Range for N: 0x0001 – 0xFFFF Time Range: 0.625 msec - 40.9 Seconds

Sniff_Min_Interval:**Size: 2 Byte**

Value	Parameter Description
N = 0xXXXX	Minimum acceptable number of Baseband slots between each sniff period. (Sniff_Max_Interval >= Sniff_Min_Interval) Length = N * 0.625 msec (1 Baseband slot) Range for N: 0x0001 – 0xFFFF Time Range: 0.625 msec - 40.9 Seconds

Sniff_Attempt:**Size: 2 Byte**

Value	Parameter Description
N = 0xXXXX	Number of Baseband slots for sniff attempt. Length = N * 0.625 msec (1 Baseband slot) Range for N: 0x0001 – 0xFFFF Time Range: 0.625 msec - 40.9 Seconds

Sniff_Timeout:**Size: 2 Byte**

Value	Parameter Description
N = 0xXXXX	Number of Baseband slots for sniff timeout. Length = N * 0.625 msec (1 Baseband slot) Range for N: 0x0001 – 0xFFFF Time Range: 0.625 msec - 40.9 Seconds

Return Parameters:

None.

Event(s) generated (unless masked away):

The Host Controller sends the Command Status event for this command to the Host when it has received the Sniff_Mode command. The Mode Change event will occur when the Sniff Mode has started for the specified connection handle. Note: no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead only the Mode Change event will indicate that this command has been completed. If an error occurs after the Command Status event has occurred, then the status in the Mode Change event will indicate the error.

4.6.3 Exit_Sniff_Mode

Command	OCF	Command Parameters	Return Parameters
HCI_Exit_Sniff_Mode	0x0004	Connection_Handle	

Description:

The Exit_Sniff_Mode command is used to end the sniff mode for a connection handle, which is currently in sniff mode. The Link Manager will determine and issue the appropriate LMP commands to remove the sniff mode for the associated Connection Handle.

Note: in addition, the connection handle cannot be one of SCO link type.

Command Parameters:

Connection_Handle:

Size: 2 Bytes (12 Bits meaningful)

Value	Parameter Description
0xXXXX	Connection Handle to be used to identify a connection. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Return Parameters:

None.

Event(s) generated (unless masked away):

A Command Status event for this command will occur when Host Controller has received the Exit_Sniff_Mode command. The Mode Change event will occur when the Sniff Mode has ended for the specified connection handle.

Note: no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, only the Mode Change event will indicate that this command has been completed.

4.6.4 Park_Mode

Command	OCF	Command Parameters	Return Parameters
HCI_Park_Mode	0x0005	Connection_Handle, Beacon_Max_Interval, Beacon_Min_Interval	

Description:

The Park Mode command is used to alter the behavior of the Link Manager, and have the LM place the baseband connection associated by the specified Connection Handle into the Park mode. The Connection_Handle command parameter is used to identify which connection is to be placed in Park mode. The Connection_Handle must be a Connection_Handle for an ACL connection. The Beacon Interval command parameters specify the acceptable length of the interval between beacons. However, the remote device may request shorter interval. The Beacon_Max_Interval parameter specifies the acceptable longest length of the interval between beacons. The Beacon_Min_Interval parameter specifies the acceptable shortest length of the interval between beacons. Therefore, the Beacon Min Interval cannot be greater than the Beacon Max Interval. The Host Controller will return the actual Beacon interval in the Interval parameter of the Mode Change event, if the command is successful. This command enables the Host to support a low-power policy for itself or several other Bluetooth devices, allows the devices to enter Inquiry Scan, Page Scan, provides support for large number of Bluetooth Devices in a single piconet, and a number of other possible activities.

Command Parameters:*Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xXXXX	Connection Handle to be used to identify a connection. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

*Beacon_Max_Interval:**Size: 2 Bytes*

Value	Parameter Description
N = 0xXXXX	Maximum acceptable number of Baseband slots between consecutive beacons. Interval Length = $N * 0.625 \text{ msec}$ (1 Baseband slot) Range for N: 0x0001 – 0xFFFF Time Range: 0.625 msec - 40.9 Seconds

Beacon_Min_Interval

Size: 2 Bytes

Value	Parameter Description
N = 0xXXXX	Minimum acceptable number of Baseband slots between consecutive beacons Interval Length = $N * 0.625 \text{ msec}$ (1 Baseband slot) Range for N: 0x0001 – 0xFFFF Time Range: 0.625 msec - 40.9 Seconds

Return Parameters:

None.

Event(s) generated (unless masked away):

The Host Controller sends the Command Status event for this command to the Host when it has received the Park_Mode command. The Mode Change event will occur when the Park Mode has started for the specified connection handle. Note: no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, only the Mode Change event will indicate that this command has been completed. If an error occurs after the Command Status event has occurred, then the status in the Mode Change event will indicate the error.

4.6.5 Exit_Park_Mod

Command	OCF	Command Parameters	Return Parameters
HCI_Exit_Park_Mode	0x0006	Connection_Handle	

Description:

The Exit_Park_Mode command is used to switch the Bluetooth device from park mode back to active mode. This command may only be issued when the device associated with the specified Connection Handle is in Park Mode. The Connection_Handle must be a Connection_Handle for an ACL connection. This function does not complete immediately.

Command Parameters:

Connection_Handle:

Size: 2 Bytes (12 Bits meaningful)

Value	Parameter Description
0xFFFF	Connection Handle to be used to identify a connection. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Return Parameters:

None.

Event(s) generated (unless masked away):

A Command Status event for this command will occur when the Host Controller has received the Exit_Park_Mode command. The Mode Change event will occur when the Park Mode has ended for the specified connection handle. Note: no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, only the Mode Change event will indicate that this command has been completed.

4.6.6 QoS_S tup

Command	OCF	Command Parameters	Return Parameters
HCI_QoS_Setup	0x0007	Connection_Handle, Flags, Service_Type, Token_Rate, Peak_Bandwidth, Latency, Delay_Variation	

Description:

The QoS_Setup command is used to specify Quality of Service parameters for a connection handle. The Connection_Handle must be a Connection_Handle for an ACL connection. These QoS parameter are the same parameters as L2CAP QoS. For more detail see "Logical Link Control and Adaptation Protocol Specification" on page 245. This allows the Link Manager to have all of the information about what the Host is requesting for each connection. The LM will determine if the QoS parameters can be met. Bluetooth devices that are both slaves and masters can use this command. When a device is a slave, this command will trigger an LMP request to the master to provide the slave with the specified QoS as determined by the LM. When a device is a master, this command is used to request a slave device to accept the specified QoS as determined by the LM of the master. The Connection_Handle command parameter is used to identify for which connection the QoS request is requested.

Command Parameters:

Connection_Handle:

Size: 2 Bytes (12 Bits meaningful)

Value	Parameter Description
0xXXXX	Connection Handle to be used to identify which connection for the QoS Setup. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Flags:

Size: 1 Byte

Value	Parameter Description
0x00 – 0xFF	Reserved for Future Use.

Service_Type:**Size: 1 Byte**

Value	Parameter Description
0x00	No Traffic.
0x01	Best Effort.
0x02	Guaranteed.
0x03-0xFF	Reserved for Future Use.

Token_Rate:**Size: 4 Bytes**

Value	Parameter Description
0xFFFFFFFF	Token Rate in bytes per second.

Peak_Bandwidth:**Size: 4 Bytes**

Value	Parameter Description
0xFFFFFFFF	Peak Bandwidth in bytes per second.

Latency:**Size: 4 Bytes**

Value	Parameter Description
0xFFFFFFFF	Latency in microseconds.

Delay_Variation:**Size: 4 Bytes**

Value	Parameter Description
0xFFFFFFFF	Delay Variation in microseconds.

Return Parameters:

None.

Event(s) generated (unless masked away):

When the Host Controller receives the QoS_Setup command, the Host Controller sends the Command Status event to the Host. When the Link Manager has completed the LMP messages to establish the requested QoS parameters, the Host Controller on the local Bluetooth device will send a QoS Setup Complete event to the Host, and the event may also be generated on the remote side if there was LMP negotiation. The values of the parameters of the QoS Setup Complete event may, however, be different on the initiating and the remote side. The QoS Setup Complete event returned by the Host Controller on the local side contains the status of this command, and returned QoS parameters describing the supported QoS for the connection.

Note: No Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, the QoS Setup Complete event will indicate that this command has been completed.

4.6.7 Role_Discovery

Command	OCF	Command Parameters	Return Parameters
HCI_Role_Discovery	0x0009	Connection_Handle	Status, Connection_Handle, Current_Role

Description:

The Role_Discovery command is used for a Bluetooth device to determine which role the device is performing for a particular Connection_Handle. The Connection_Handle must be a Connection_Handle for an ACL connection.

Command Parameters:

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xXXXX	Connection_Handle to be used to identify a connection. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Return Parameters:

Status: *Size: 1 Byte*

Value	Parameter Description
0x00	Role_Discovery command succeeded,
0x01-0xFF	Role_Discovery command failed. See Table 6.1 on page 745 for list of Error Codes.

Connection_Handle: *Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xXXXX	Connection_Handle to be used to identify which connection the Role_Discovery command was issued on. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Current_Role: *Size: 1 Byte*

Value	Parameter Description
0x00	Current Role is Master for this Connection_Handle.
0x01	Current Role is Slave for this Connection_Handle.

Event(s) generated (unless masked away):

When the Role_Discovery command has completed, a Command Complete event will be generated.

4.6.8 Switch_Role

Command	OCF	Command Parameters	Return Parameters
HCI_Switch_Role	0x000B	BD_ADDR, Role	

Description:

The Switch_Role command is used for a Bluetooth device to switch the current role the device is performing for a particular connection with another specified Bluetooth device. The BD_ADDR command parameter indicates for which connection the role switch is to be performed. The Role indicates the requested new role that the local device performs.

Note: the BD_ADDR command parameter must specify a Bluetooth device for which a connection already exists.

Command Parameters:**BD_ADDR:***Size: 6 Bytes*

Value	Parameter Description
0XXXXXXXXXX XX	BD_ADDR for the connected device with which a role switch is to be performed.

Role:*Size: 1 Byte*

Value	Parameter Description
0x00	Change own Role to Master for this BD_ADDR.
0x01	Change own Role to Slave for this BD_ADDR.

Return Parameters:

None.

Event(s) generated (unless masked away):

A Command Status event for this command will occur when the Host Controller has received the Switch_Role command. When the role switch is performed, a Role Change event will occur to indicate that the roles have been changed, and will be communicated to both Hosts.

Note: no Command Complete event will be sent by the Host Controller to indicate that this command has been completed. Instead, only the Role Change event will indicate that this command has been completed.

4.6.9 Read_Link_Policy_Settings

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Link_Policy_Settings	0x000C	Connection_Handle	Status, Connection_Handle Link_Policy_Settings

Description:

This command will read the Link Policy setting for the specified Connection Handle. The Link_Policy_Settings parameter determines the behavior of the local Link Manager when it receives a request from a remote device or it determines itself to change the master-slave role or to enter the hold, sniff, or park mode. The local Link Manager will automatically accept or reject such a request from the remote device, and may even autonomously request itself, depending on the value of the Link_Policy_Settings parameter for the corresponding Connection_Handle. When the value of the Link_Policy_Settings parameter is changed for a certain Connection_Handle, the new value will only be used for requests from a remote device or from the local Link Manager itself made after this command has been completed. The Connection_Handle must be a Connection_Handle for an ACL connection. By enabling each mode individually, the Host can choose any combination needed to support various modes of operation. Multiple LM policies may be specified for the Link_Policy_Settings parameter by performing a bitwise OR operation of the different activity types.

Command Parameters:

Connection_Handle:

Size: 2 Bytes (12 Bits meaningful)

Value	Parameter Description
0xXXXX	Connection Handle to be used to identify a connection. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_Link_Policy_Settings command succeeded.
0x01-0xFF	Read_Link_Policy_Settings command failed. See Table 6.1 on page 745 for list of Error Codes.

Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)**

Value	Parameter Description
0XXXXX	Connection Handle to be used to identify a connection. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Link_Policy_Settings**Size: 2 Bytes**

Value	Parameter Description
0x0000	Disable All LM Modes.
0x0001	Enable Master Slave Switch.
0x0002	Enable Hold Mode.
0x0004	Enable Sniff Mode.
0x0008	Enable Park Mode.
0x0010	Reserved for Future Use.
—	
0x8000	

Event(s) generated (unless masked away):

When the Read_Link_Policy_Settings command has completed, a Command Complete event will be generated.

4.6.10 Write_Link_Policy_Settings

Command	OCF	Command Parameters	Return Parameters
HCI_Write_Link_Policy_Settings	0x000D	Connection_Handle, Link_Policy_Settings	Status, Connection_Handle

Description:

This command will write the Link Policy setting for the specified Connection Handle. The Link_Policy_Settings parameter determines the behavior of the local Link Manager when it receives a request from a remote device or it determines itself to change the master-slave role or to enter the hold, sniff, or park mode. The local Link Manager will automatically accept or reject such a request from the remote device, and may even autonomously request itself, depending on the value of the Link_Policy_Settings parameter for the corresponding Connection_Handle. When the value of the Link_Policy_Settings parameter is changed for a certain Connection_Handle, the new value will only be used for requests from a remote device or from the local Link Manager itself made after this command has been completed. The Connection_Handle must be a Connection_Handle for an ACL connection. By enabling each mode individually, the Host can choose any combination needed to support various modes of operation. Multiple LM policies may be specified for the Link_Policy_Settings parameter by performing a bitwise OR operation of the different activity types.

Command Parameters:

Connection_Handle:

Size: 2 Bytes (12 Bits meaningful)

Value	Parameter Description
0xFFFF	Connection Handle to be used to identify a connection. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Link_Policy_Settings**Size: 2 Bytes**

Value	Parameter Description
0x0000	Disable All LM Modes D fault.
0x0001	Enable Master Slave Switch.
0x0002	Enable Hold Mode.
0x0004	Enable Sniff Mode.
0x0008	Enable Park Mode.
0x0010	Reserved for Future Use.
0x8000	

Return Parameters:**Status:****Size: 1 Byte**

Value	Parameter Description
0x00	Write_Link_Policy_Settings command succeeded.
0x01-0xFF	Write_Link_Policy_Settings command failed. See Table 6.1 on page 745 for list of Error Codes.

Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)**

Value	Parameter Description
0xFFFF	Connection Handle to be used to identify a connection. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Event(s) generated (unless masked away):

When the Write_Link_Policy_Settings command has completed, a Command Complete event will be generated.

4.7 HOST CONTROLLER & BASEBAND COMMANDS

The Host Controller & Baseband Commands provide access and control to various capabilities of the Bluetooth hardware. These parameters provide control of Bluetooth devices and of the capabilities of the Host Controller, Link Manager, and Baseband. The host device can use these commands to modify the behavior of the local device. For the HCI Control and Baseband Commands, the OGF is defined as 0x03

Command	Command Summary Description
Set_Event_Mask	The Set_Event_Mask command is used to control which events are generated by the HCI for the Host.
Reset	The Reset command will reset the Bluetooth Host Controller, Link Manager, and the radio module.
Set_Event_Filter	The Set_Event_Filter command is used by the Host to specify different event filters. The Host may issue this command multiple times to request various conditions for the same type of event filter and for different types of event filters.
Flush	The Flush command is used to discard all data that is currently pending for transmission in the Host Controller for the specified connection handle, even if there currently are chunks of data that belong to more than one L2CAP packet in the Host Controller.
Read_PIN_Type	The Read_PIN_Type command is used for the Host to read the value that is specified to indicate whether the Host supports variable PIN or only fixed PINs.
Write_PIN_Type	The Write_PIN_Type command is used for the Host to specify whether the Host supports variable PIN or only fixed PINs.
Create_New_Unit_Key	The Create_New_Unit_Key command is used to create a new unit key.
Read_Stored_Link_Key	The Read_Stored_Link_Key command provides the ability to read one or more link keys stored in the Bluetooth Host Controller.
Write_Stored_Link_Key	The Write_Stored_Link_Key command provides the ability to write one or more link keys to be stored in the Bluetooth Host Controller.

Command	Command Summary Description
Delete_Stored_Link_Key	The Delete_Stored_Link_Key command provides the ability to remove one or more of the link keys stored in the Bluetooth Host Controller.
Change_Local_Name	The Change_Local_Name command provides the ability to modify the user-friendly name for the Bluetooth device.
Read_Local_Name	The Read_Local_Name command provides the ability to read the stored user-friendly name for the Bluetooth device.
Read_Connection_Accept_Timeout	The Read_Connection_Accept_Timeout command will read the value for the Connection_Accept_Timeout configuration parameter, which allows the Bluetooth hardware to automatically deny a connection request after a specified period has occurred, and to refuse a new connection.
Write_Connection_Accept_Timeout	The Write_Connection_Accept_Timeout will write the value for the Connection_Accept_Timeout configuration parameter, which allows the Bluetooth hardware to automatically deny a connection request after a specified period has occurred, and to refuse a new connection.
Read_Page_Timeout	The Read_Page_Timeout command will read the value for the Page_Reply_Timeout configuration parameter, which allows the Bluetooth hardware to define the amount of time a connection request will wait for the remote device to respond before the local device returns a connection failure.
Write_Page_Timeout	The Write_Page_Timeout command will write the value for the Page_Reply_Timeout configuration parameter, which allows the Bluetooth hardware to define the amount of time a connection request will wait for the remote device to respond before the local device returns a connection failure.
Read_Scan_Enable	The Read_Scan_Enable command will read the value for the Scan_Enable configuration parameter, which controls whether or not the Bluetooth device will periodically scan for page attempts and/or inquiry requests from other Bluetooth devices.

Command	Command Summary Description
Write_Scan_Enable	The Write_Scan_Enable command will write the value for the Scan_Enable configuration parameter, which controls whether or not the Bluetooth device will periodically scan for page attempts and/or inquiry requests from other Bluetooth devices.
Read_Page_Scan_Activity	The Read_Page_Scan_Activity command will read the values for the Page_Scan_Interval and Page_Scan_Window configuration parameters. Page_Scan_Interval defines the amount of time between consecutive page scans. Page_Scan_Window defines the duration of the page scan.
Write_Page_Scan_Activity	The Write_Page_Scan_Activity command will write the value for Page_Scan_Interval and Page_Scan_Window configuration parameters. Page_Scan_Interval defines the amount of time between consecutive page scans. Page_Scan_Window defines the duration of the page scan.
Read_Inquiry_Scan_Activity	The Read_Inquiry_Scan_Activity command will read the value for Inquiry_Scan_Interval and Inquiry_Scan_Window configuration parameters. Inquiry_Scan_Interval defines the amount of time between consecutive inquiry scans. Inquiry_Scan_Window defines the amount of time for the duration of the inquiry scan.
Write_Inquiry_Scan_Activity	The Write_Inquiry_Scan_Activity command will write the value for Inquiry_Scan_Interval and Inquiry_Scan_Window configuration parameters. Inquiry_Scan_Interval defines the amount of time between consecutive inquiry scans. Inquiry_Scan_Window defines the amount of time for the duration of the inquiry scan.
Read_Authentication_Enable	The Read_Authentication_Enable command will read the value for the Authentication_Enable parameter, which controls whether the Bluetooth device will require authentication for each connection with other Bluetooth devices.
Write_Authentication_Enable	The Write_Authentication_Enable command will write the value for the Authentication_Enable parameter, which controls whether the Bluetooth device will require authentication for each connection with other Bluetooth devices.
Read_Encryption_Mode	The Read_Encryption_Mode command will read the value for the Encryption_Mode parameter, which controls whether the Bluetooth device will require encryption for each connection with other Bluetooth devices.

Command	Command Summary Description
Write_Encryption_Mode	The Write_Encryption_Mode command will write the value for the Encryption_Mod parameter, which controls whether the Bluetooth device will require encryption for each connection with other Bluetooth devices.
Read_Class_of_Device	The Read_Class_of_Device command will read the value for the Class_of_Device parameter, which is used to indicate its capabilities to other devices.
Write_Class_of_Device	The Write_Class_of_Device command will write the value for the Class_of_Device parameter, which is used to indicate its capabilities to other devices.
Read_Voice_Setting	The Read_Voice_Setting command will read the values for the Voice_Setting parameter, which controls all the various settings for the voice connections.
Write_Voice_Setting	The Write_Voice_Setting command will write the values for the Voice_Setting parameter, which controls all the various settings for the voice connections.
Read_Automatic_Flush_Timeout	The Read_Automatic_Flush_Timeout will read the value for the Flush_Timeout parameter for the specified connection handle. The Flush_Timeout parameter is only used for ACL connections.
Write_Automatic_Flush_Timeout	The Write_Automatic_Flush_Timeout will write the value for the Flush_Timeout parameter for the specified connection handle. The Flush_Timeout parameter is only used for ACL connections.
Read_Num_Broadcast_Retransmissions	The Read_Num_Broadcast_Retransmissions command will read the parameter value for the Number of Broadcast Retransmissions for the device. Broadcast packets are not acknowledged and are unreliable. This parameter is used to increase the reliability of a broadcast message by retransmitting the broadcast message multiple times.
Write_Num_Broadcast_Retransmissions	The Write_Num_Broadcast_Retransmissions command will write the parameter value for the Number of Broadcast Retransmissions for the device. Broadcast packets are not acknowledged and are unreliable. This parameter is used to increase the reliability of a broadcast message by retransmitting the broadcast message multiple times.

Command	Command Summary Description
Read_Hold_Mode_Activity	The Read_Hold_Mode_Activity command will read the value for the Hold_Mode_Activity parameter. This value is used to determine what activity the device should do when it is in hold mode.
Write_Hold_Mode_Activity	The Write_Hold_Mode_Activity command will write the value for the Hold_Mode_Activity parameter. This value is used to determine what activity the device should do when it is in hold mode.
Read_Transmit_Power_Level	The Read_Transmit_Power_Level command will read the values for the Transmit_Power_Level parameter for the specified Connection Handle.
Read_SCO_Flow_Control_Enable	The Read_SCO_Flow_Control_Enable command provides the ability to read the SCO_Flow_Control_Enable setting. By using this setting, the Host can decide if the Host Controller will send Number Of Completed Packets events for SCO Connection Handles.
Write_SCO_Flow_Control_Enable	The Write_SCO_Flow_Control_Enable command provides the ability to write the SCO_Flow_Control_Enable setting. By using this setting, the Host can decide if the Host Controller will send Number Of Completed Packets events for SCO Connection Handles.
Set_Host_Controller_To_Host_Flow_Control	The Set_Host_Controller_To_Host_Flow_Control command is used by the Host to turn flow control on or off in the direction from the Host Controller to the Host.
Host_Buffer_Size	The Host_Buffer_Size command is used by the Host to notify the Host Controller about its buffer sizes for ACL and SCO data. The Host Controller will segment the data to be transmitted from the Host Controller to the Host, so that data contained in HCI Data Packets will not exceed these sizes.
Host_Number_Of_Completed_Packets	The Host_Number_Of_Completed_Packets command is used by the Host to indicate to the Host Controller when the Host is ready to receive more HCI packets for any connection handle.
Read_Link_Supervision_Timeout	The Read_Link_Supervision_Timeout command will read the value for the Link_Supervision_Timeout parameter for the device. This parameter is used by the master or slave Bluetooth device to monitor link loss.

Command	Command Summary Description
Write_Link_Supervision_Timeout	The Write_Link_Supervision_Timeout command will write the value for the Link_Supervision_Timeout parameter for the device. This parameter is used by the master or slave Bluetooth device to monitor link loss.
Read_Number_Of_Supported_IAC	The Read_Number_Of_Supported_IAC command will read the value for the number of Inquiry Access Codes (IAC) that the local Bluetooth device can simultaneously listen for during an Inquiry Scan.
Read_Current_IAC_LAP	The Read_Current_IAC_LAP command will read the LAP(s) used to create the Inquiry Access Codes (IAC) that the local Bluetooth device is simultaneously scanning for during Inquiry Scans.
Write_Current_IAC_LAP	The Write_Current_IAC_LAP will write the LAP(s) used to create the Inquiry Access Codes (IAC) that the local Bluetooth device is simultaneously scanning for during Inquiry Scans.
Read_Page_Scan_Period_Mode	The Read_Page_Scan_Period_Mode command is used to read the mandatory Page_Scan_Period_Mode of the local Bluetooth device.
Write_Page_Scan_Period_Mode	The Write_Page_Scan_Period_Mode command is used to write the mandatory Page_Scan_Period_Mode of the local Bluetooth device.
Read_Page_Scan_Mode	The Read_Page_Scan_Mode command is used to read the default Page_Scan_Mode of the local Bluetooth device.
Write_Page_Scan_Mode	The Write_Page_Scan_Mode command is used to write the default Page_Scan_Mode of the local Bluetooth device.

4.7.1 S t_Ev nt_Mask

Command	OCF	Command Parameters	Return Parameters
HCI_Set_Event_Mask	0x0001	Event_Mask	Status

Description:

The Set_Event_Mask command is used to control which events are generated by the HCI for the Host. If the bit in the Event_Mask is set to a one, then the event associated with that bit will be enabled. The Host has to deal with each event that occurs by the Bluetooth devices. The event mask allows the Host to control how much it is interrupted.

Note: the Command Complete event, Command Status event and Number Of Completed Packets event cannot be masked. These events always occur. The Event_Mask is a bit mask of all of the events specified in Table 5.1 on page 703.

Command Parameters:**Event_Mask:***Size: 8 Bytes*

Value	Parameter Description
0x0000000000000000	No events specified
0x0000000000000001	Inquiry Complete event
0x0000000000000002	Inquiry Result event
0x0000000000000004	Connection Complete event
0x0000000000000008	Connection Request event
0x0000000000000010	Disconnection Complete event
0x0000000000000020	Authentication Complete event
0x0000000000000040	Remote Name Request Complete event
0x0000000000000080	Encryption Change event
0x0000000000000100	Change Connection Link Key Complete event
0x0000000000000200	Master Link Key Complete event
0x0000000000000400	Read Remote Supported Features Complete event
0x0000000000000800	Read Remote Version Information Complete event
0x0000000000001000	QoS Setup Complete event
0x0000000000002000	Command Complete event
0x0000000000004000	Command Status event

0x0000000000008000	Hardware Error event
0x0000000000010000	Flush Occurred event
0x0000000000020000	Role Change event

Value	Parameter Description
0x0000000000040000	Number Of Completed Packets event
0x0000000000080000	Mode Change event
0x0000000000100000	Return Link Keys event
0x0000000000200000	PIN Code Request event
0x0000000000400000	Link Key Request event
0x0000000000800000	Link Key Notification event
0x0000000001000000	Loopback Command event
0x0000000002000000	Data Buffer Overflow event
0x0000000004000000	Max Slots Change event
0x0000000008000000	Read Clock Offset Complete event
0x0000000010000000	Connection Packet Type Changed event
0x0000000020000000	QoS Violation event
0x0000000040000000	Page Scan Mode Change event
0x0000000080000000	Page Scan Repetition Mode Change event
0x0000000100000000 to 0x8000000000000000	Reserved for future use
0x00000000FFFFFFFF	Default (All events enabled)

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Set_Event_Mask command succeeded.
0x01-0xFF	Set_Event_Mask command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Set_Event_Mask command has completed, a Command Complete event will be generated.

4.7.2 Reset

Command	OCF	Command Parameters	Return Parameters
HCI_Reset	0x0003		Status

Description:

The Reset command will reset the Bluetooth Host Controller, Link Manager, and the radio module. The current operational state will be lost, and all queued packets will be lost. After the reset is completed, the Bluetooth device will enter standby mode.

Note: after the reset has completed, the Host Controller will automatically revert to the default values for the parameters for which default values are defined in this specification.

Command Parameters:

None.

Return Parameters:**Status:***Size: 1 Byte*

Value	Parameter Description
0x00	Reset command succeeded, was received and will be executed.
0x01-0xFF	Reset command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

Before the Reset command will be executed, a Command Complete event needs to be returned to indicate to the Host that the Reset command was received and will be executed.

4.7.3 Set_Event_Filter

Command	OCF	Command Parameters	Return Parameters
HCI_Set_Event_Filter	0x0005	Filter_Type, Filter_Condition_Type, Condition	Status

Description:

The Set_Event_Filter command is used by the Host to specify different event filters. The Host may issue this command multiple times to request various conditions for the same type of event filter and for different types of event filters. The event filters are used by the Host to specify items of interest, which allow the Host Controller to send only events which interest the Host. Only some of the events have event filters. By default (before this command has been issued after power-on or Reset) no filters are set, and the Auto_Accept_Flag is off (incoming connections are not automatically accepted). An event filter is added each time this command is sent from the Host and the Filter_Condition_Type is not equal to 0x00. (The old event filters will not be overwritten). To clear all event filters, the Filter_Type = 0x00 is used. The Auto_Accept_Flag will then be set to off.

To clear event filters for only a certain Filter_Type, the Filter_Condition_Type = 0x00 is used. The Inquiry Result filter allows the Host Controller to filter out Inquiry Result events. The Inquiry Result filter allows the Host to specify that the Host Controller only sends Inquiry Results to the Host if the Inquiry Result event meets one of the specified conditions set by the Host. For the Inquiry Result filter, the Host can specify one or more of the following Filter Condition Types:

1. A new device responded to the Inquiry process
2. A device with a specific Class of Device responded to the Inquiry process
3. A device with a specific BD_ADDR responded to the Inquiry process

The Inquiry Result filter is used in conjunction with the Inquiry and Periodic Inquiry command. The Connection Setup filter allows the Host to specify that the Host Controller only sends a Connection Complete or Connection Request event to the Host if the event meets one of the specified conditions set by the Host. For the Connection Setup filter, the Host can specify one or more of the following Filter Condition Types:

1. Allow Connections from all devices
2. Allow Connections from a device with a specific Class of Device
3. Allow Connections from a device with a specific BD_ADDR

For each of these conditions, an `Auto_Accept_Flag` parameter allows the Host to specify what action should be done when the condition is met. The `Auto_Accept_Flag` allows the Host to specify if the incoming connection should be auto accepted (in which case the Host Controller will send the `Connection Complete` event to the Host when the connection is completed) or if the Host should make the decision (in which case the Host Controller will send the `Connection Request` event to the Host, to elicit a decision on the connection).

The `Connection Setup` filter is used in conjunction with the `Read_Write_Scan_Enable` commands. If the local device is in the process of a page scan, and is paged by another device which meets one on the conditions set by the Host, and the `Auto_Accept_Flag` is off for this device, then a `Connection Request` event will be sent to the Host by the Host Controller. A `Connection Complete` event will be sent later on after the Host has responded to the incoming connection attempt. In this same example, if the `Auto_Accept_Flag` is on, then a `Connection Complete` event will be sent to the Host by the Host Controller. (No `Connection Request` event will be sent in that case.)

The Host Controller will store these filters in volatile memory until the Host clears the event filters using the `Set_Event_Filter` command or until the `Reset` command is issued. The number of event filters the Host Controller can store is implementation dependent. If the Host tries to set more filters than the Host Controller can store, the Host Controller will return the "Memory Full" error code and the filter will not be installed.

Note: the `Clear All Filters` has no Filter Condition Types or Conditions.

Note: In the condition that a connection is auto accepted, a `Link Key Request` event and possibly also a `PIN Code Request` event and a `Link Key Notification` event could be sent to the Host by the Host Controller before the `Connection Complete` event is sent.

If there is a contradiction between event filters, the latest set event filter will override older ones. An example is an incoming connection attempt where more than one `Connection Setup` filter matches the incoming connection attempt, but the `Auto-Accept_Flag` has different values in the different filters.

Command Parameters:

Filter_Type:

Size: 1 Byte

Value	Parameter Description
0x00	Clear All Filters (Note: In this case, the Filter_Condition_type and Condition parameters should not be given, they should have a length of 0 bytes. Filter_Type should be the only parameter.)
0x01	Inquiry Result
0x02	Connection Setup.
0x03-0xFF	Reserved for Future Use.

Filter Condition Types: For each Filter Type one or more Filter Condition types exists.

Inquiry_Result_Filter_Condition_Type:

Size: 1 Byte

Value	Parameter Description
0x00	A new device responded to the Inquiry process. (Note: A device may be reported to the Host in an Inquiry Result event more than once during an inquiry or inquiry period depending on the implementation, see description in Section 4.5.1 on page 542 and Section 4.5.3 on page 545)
0x01	A device with a specific Class of Device responded to the Inquiry process.
0x02	A device with a specific BD_ADDR responded to the Inquiry process.
0x03-0xFF	Reserved for Future Use

Connection_Setup_Filter_Condition_Type:

Size: 1 Byte

Value	Parameter Description
0x00	Allow Connections from all devices.
0x01	Allow Connections from a device with a specific Class of Device.
0x02	Allow Connections from a device with a specific BD_ADDR.
0x03-0xFF	Reserved for Future Use.

Condition: For each Filter Condition Type defined for the Inquiry Result Filter and the Connection Setup Filter, zero or more Condition parameters are required – depending on the filter condition type and filter type.

Condition for Inquiry_Result_Filter_Condition_Type = 0x00

Condition:

Size: 0 Byte

Value	Parameter Description
	The Condition parameter is not used.

Condition for Inquiry_Result_Filter_Condition_Type = 0x01

Condition:

Size: 6 Bytes

Class_of_Device:

Size: 3 Bytes

Value	Parameter Description
0x000000	Default, Return All Devices.
0xxxxxxx	Class of Device of Interest.

Class_of_Device_Mask:

Size: 3 Bytes

Value	Parameter Description
0xxxxxxx	Bit Mask used to determine which bits of the Class of Device parameter are 'don't care'. Zero-value bits in the mask indicate the 'don't care' bits of the Class of Device.

Condition for Inquiry_Result_Filter_Condition_Type = 0x02

Condition:

Size: 6 Bytes

BD_ADDR:

Size: 6 Bytes

Value	Parameter Description
0xxxxxxxxxxx xx	BD_ADDR of the Device of Interest

Condition for Connection_Setup_Filter_Condition_Type = 0x00

Condition:

Size: 1 Byte

Auto_Accept_Flag:

Size: 1 Byte

Value	Parameter Description
0x01	Do NOT Auto accept the connection.
0x02	Do Auto accept the connection.
0x03 – 0xFF	Reserved for future use.

Condition for Connection_Setup_Filter_Condition_Type = 0x01

Condition:

Size: 7 Bytes

Class_of_Device:

Size: 3 Bytes

Value	Parameter Description
0x000000	Default, Return All Devices.
0xxxxxxx	Class of Device of Interest.

Class_of_Device_Mask:

Size: 3 Bytes

Value	Parameter Description
0xxxxxxx	Bit Mask used to determine which bits of the Class of Device parameter are 'don't care'. Zero-value bits in the mask indicate the 'don't care' bits of the Class of Device.

Auto_Accept_Flag:

Size: 1 Byte

Value	Parameter Description
0x01	Do NOT Auto accept the connection.
0x02	Do Auto accept the connection.
0x03 – 0xFF	Reserved for future use.

Condition for Connection_Setup_Filter_Condition_Type = 0x02

Condition:

Size: 7 Bytes

BD_ADDR:

Size: 6 Bytes

Value	Parameter Description
0xxxxxxxxxxx xx	BD_ADDR of the Device of Interest.

Auto_Accept_Flag:

Size: 1 Byte

Value	Parameter Description
0x01	Do NOT Auto accept the connection.
0x02	Do Auto accept the connection.
0x03 – 0xFF	Reserved for future use.

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Set_Event_Filter command succeeded.
0x01-0xFF	Set_Event_Filter command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

A Command Complete event for this command will occur when the Host Controller has enabled the filtering of events. When one of the conditions are met, a specific event will occur.

4.7.4 Flush

Command	OCF	Command Parameters	Return Parameters
HCI_Flush	0x0008	Connection_Handle	Status, Connection_Handle

Description:

The Flush command is used to discard all data that is currently pending for transmission in the Host Controller for the specified connection handle, even if there currently are chunks of data that belong to more than one L2CAP packet in the Host Controller. After this, all data that is sent to the Host Controller for the same connection handle will be discarded by the Host Controller until an HCI Data Packet with the start Packet_Boundary_Flag (0x02) is received. When this happens, a new transmission attempt can be made. This command will allow higher-level software to control how long the baseband should try to retransmit a baseband packet for a connection handle before all data that is currently pending for transmission in the Host Controller should be flushed. Note that the Flush command is used for ACL connections ONLY. In addition to the Flush command, the automatic flush timers (see section 4.7.31 on page 647) can be used to automatically flush the L2CAP packet that is currently being transmitted after the specified flush timer has expired.

Command Parameters:*Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xXXXX	Connection Handle to be used to identify which connection to flush. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Flush command succeeded.
0x01-0xFF	Flush command failed. See Table 6.1 on page 745 for list of Error Codes.

*Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xXXXX	Connection Handle to be used to identify which connection the flush command was issued on. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Event(s) generated (unless masked away):

The Flush Occurred event will occur once the flush is completed. A Flush Occurred event could be from an automatic Flush or could be caused by the Host issuing the Flush command. When the Flush command has completed, a Command Complete event will be generated, to indicate that the Host caused the Flush.

4.7.5 Read_PIN_Typ

Command	OCF	Command Parameters	Return Parameters
HCI_Read_PIN_Type	0x0009		Status, PIN_Type

Description:

The Read_PIN_Type command is used for the Host to read whether the Link Manager assumes that the Host supports variable PIN codes only a fixed PIN code. The Bluetooth hardware uses the PIN-type information during pairing.

Command Parameters:

None.

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_PIN_Type command succeeded.
0x01-0xFF	Read_PIN_Type command failed. See Table 6.1 on page 745 for list of Error Codes.

PIN_Type:

Size: 1 Byte

Value	Parameter Description
0x00	Variable PIN.
0x01	Fixed PIN.

Event(s) generated (unless masked away):

When the Read_PIN_Type command has completed, a Command Complete event will be generated.

4.7.6 Write_PIN_Type

Command	OCF	Command Parameters	Return Parameters
HCI_Write_PIN_Type	0x000A	PIN_Type	Status

Description:

The Write_PIN_Type command is used for the Host to write to the Host Controller whether the Host supports variable PIN codes or only a fixed PIN code. The Bluetooth hardware uses the PIN-type information during pairing.

Command Parameters:*PIN_Type:**Size: 1 Byte*

Value	Parameter Description
0x00	Variable PIN.
0x01	Fixed PIN.

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Write PIN Type command succeeded.
0x01-0xFF	Write PIN Type command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_PIN_Type command has completed, a Command Complete event will be generated.

4.7.7 Create_New_Unit_Key

Command	OCF	Command Parameters	Return Parameters
HCI_Create_New_Unit_Key	0x000B		Status

Description:

The Create_New_Unit_Key command is used to create a new unit key. The Bluetooth hardware will generate a random seed that will be used to generate the new unit key. All new connection will use the new unit key, but the old unit key will still be used for all current connections.

Note: this command will not have any effect for a device which doesn't use unit keys (i.e. a device which uses only combination keys).

Command Parameters:

None.

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Create New Unit Key command succeeded.
0x01-0xFF	Create New Unit Key command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Create_New_Unit_Key command has completed, a Command Complete event will be generated.

4.7.8 Read_Stored_Link_Key

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Stored_Link_Key	0x000D	BD_ADDR, Read_All_Flag	Status, Max_Num_Keys, Num_Keys_Read

Description:

The Read_Stored_Link_Key command provides the ability to read one or more link keys stored in the Bluetooth Host Controller. The Bluetooth Host Controller can store a limited number of link keys for other Bluetooth devices. Link keys are shared between two Bluetooth devices, and are used for all security transactions between the two devices. A Host device may have additional storage capabilities, which can be used to save additional link keys to be reloaded to the Bluetooth Host Controller when needed. The Read_All_Flag parameter is used to indicate if all of the stored Link Keys should be returned. If Read_All_Flag indicates that all Link Keys are to be returned, then the BD_ADDR command parameter must be ignored. The BD_ADDR command parameter is used to identify which link key to read. The stored Link Keys are returned by one or more Return Link Keys events.

Command Parameters:**BD_ADDR:***Size: 6 Bytes*

Value	Parameter Description
0xxxxxxxxxxxxx	BD_ADDR for the stored link key to be read.

Read_All_Flag:*Size: 1 Byte*

Value	Parameter Description
0x00	Return Link Key for specified BD_ADDR.
0x01	Return all stored Link Keys.
0x02-0xFF	Reserved for future use.

Return Parameters:**Status:***Size: 1 Byte*

Value	Parameter Description
0x00	Read_Stored_Link_Key command succeeded.
0x01-0xFF	Read_Stored_Link_Key command failed. See Table 6.1 on page 745 for list of Error Codes.

Max_Num_Keys:**Size: 2 Byte**

Value	Parameter Description
0xFFFF	Maximum Number of Link Keys which the Host Controller can store. Range: 0x0000 – 0xFFFF

Num_Keys_Read:**Size: 2 Bytes**

Value	Parameter Description
0xFFFF	Number of Link Keys Read. Range: 0x0000 – 0xFFFF

Event(s) generated (unless masked away):

Zero or more instances of the Return Link Keys event will occur after the command is issued. When there are no link keys stored, no Return Link Keys events will be returned. When there are link keys stored, the number of link keys returned in each Return Link Keys event is implementation specific. When the Read Stored Link Key command has completed a Command Complete event will be generated.

4.7.9 Write_Stored_Link_Key

Command	OCF	Command Parameters	Return Parameters
HCI_Write_Stored_Link_Key	0x0011	Num_Keys_To_Write, BD_ADDR[i], Link_Key[i]	Status, Num_Keys_Written

Description:

The Write_Stored_Link_Key command provides the ability to write one or more link keys to be stored in the Bluetooth Host Controller. The Bluetooth Host Controller can store a limited number of link keys for other Bluetooth devices. If no additional space is available in the Bluetooth Host Controller then no additional link keys will be stored. If space is limited and if all the link keys to be stored will not fit in the limited space, then the order of the list of link keys without any error will determine which link keys are stored. Link keys at the beginning of the list will be stored first. The Num_Keys_Written parameter will return the number of link keys that were successfully stored. If no additional space is available, then the Host must delete one or more stored link keys before any additional link keys are stored. The link key replacement algorithm is implemented by the Host and not the Host Controller. Link keys are shared between two Bluetooth devices and are used for all security transactions between the two devices. A Host device may have additional storage capabilities, which can be used to save additional link keys to be reloaded to the Bluetooth Host Controller when needed.

Note: Link Keys are only stored by issuing this command.

Command Parameters:

Num_Keys_To_Write:

Size: 1 Byte

Value	Parameter Description
0xXX	Number of Link Keys to Write.

BD_ADDR [i]:

*Size: 6 Bytes * Num_Keys_To_Write*

Value	Parameter Description
0XXXXXXXXXXXXX	BD_ADDR for the associated Link Key.

Link_Key[i]:

*Size: 16 Bytes * Num_Keys_To_Write*

Value	Parameter Description
0XXXXXXXXXXXXX XXXXXXXXXXXXX XXXXXXXXXXXXX	Link Key for the associated BD_ADDR.

R turn Parameters:**Status:****Size: 1 Byte**

Value	Parameter Description
0x00	Write_Stored_Link_Key command succeeded.
0x01-0xFF	Write_Stored_Link_Key command failed. See Table 6.1 on page 745 for list of Error Codes.

Num_Keys_Written:**Size: 1 Bytes**

Value	Parameter Description
0xXX	Number of Link Keys successfully written. Range: 0x00 – 0xFF

Event(s) generated (unless masked away):

When the Write_Stored_Link_Key command has completed, a Command Complete event will be generated.

4.7.10 Delete Stored Link Key

Command	OCF	Command Parameters	Return Parameters
HCI_Delete_Stored_Link_Key	0x0012	BD_ADDR, Delete_All_Flag	Status, Num_Keys_Deleted

Description:

The Delete_Stored_Link_Key command provides the ability to remove one or more of the link keys stored in the Bluetooth Host Controller. The Bluetooth Host Controller can store a limited number of link keys for other Bluetooth devices. Link keys are shared between two Bluetooth devices and are used for all security transactions between the two devices. The Delete_All_Flag parameter is used to indicate if all of the stored Link Keys should be deleted. If the Delete_All_Flag indicates that all Link Keys are to be deleted, then the BD_ADDR command parameter must be ignored. This command provides the ability to negate all security agreements between two devices. The BD_ADDR command parameter is used to identify which link key to delete. If a link key is currently in use for a connection, then the link key will be deleted when all of the connections are disconnected.

Command Parameters:**BD_ADDR:***Size: 6 Bytes*

Value	Parameter Description
0xFFFFFFFFXXXX	BD_ADDR for the link key to be deleted.

Delete_All_Flag:*Size: 1 Byte*

Value	Parameter Description
0x00	Delete only the Link Key for specified BD_ADDR.
0x01	Delete all stored Link Keys.
0x02-0xFF	Reserved for future use.

Return Parameters:**Status:***Size: 1 Byte*

Value	Parameter Description
0x00	Delete_Stored_Link_Key command succeeded.
0x01-0xFF	Delete_Stored_Link_Key command failed. See Table 6.1 on page 745 for list of Error Codes.

Num_Keys_Deleted:**Size: 2 Bytes**

Value	Parameter Description
0xFFFF	Number of Link Keys Deleted

Event(s) generated (unless masked away):

When the Delete_Stored_Link_Key command has completed, a Command Complete event will be generated.

4.7.11 Change_Local_Name

Command	OCF	Command Parameters	Return Parameters
HCI_Change_Local_Name	0x0013	Name	Status

Description:

The Change_Local_Name command provides the ability to modify the user-friendly name for the Bluetooth device. A Bluetooth device may send a request to get the user-friendly name of another Bluetooth device. The user-friendly name provides the user with the ability to distinguish one Bluetooth device from another. The Name command parameter is a UTF-8 encoded string with up to 248 bytes in length. The Name command parameter should be null-terminated (0x00) if the UTF-8 encoded string is less than 248 bytes.

Note: the Name Parameter is transmitted starting with the first byte of the name. This is an exception to the Little Endian order format for transmitting multi-byte parameters.

Command Parameters:*Name:**Size: 248 Bytes*

Value	Parameter Description
	A UTF-8 encoded User-Friendly Descriptive Name for the device. The UTF-8 encoded Name can be up to 248 bytes in length. If it is shorter than 248 bytes, the end is indicated by a NULL byte (0x00).
	Null terminated Zero length String. Default.

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Change_Local_Name command succeeded.
0x01-0xFF	Change_Local_Name command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Change_Local_Name command has completed, a Command Complete event will be generated.

4.7.12 Read_Local_Name

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Local_Name	0x0014		Status, Name

Description:

The Read_Local_Name command provides the ability to read the stored user-friendly name for the Bluetooth device. The user-friendly name provides the user the ability to distinguish one Bluetooth device from another. The Name return parameter is a UTF-8 encoded string with up to 248 bytes in length. The Name return parameter will be null terminated (0x00) if the UTF-8 encoded string is less than 248 bytes.

Note: the Name Parameter is transmitted starting with the first byte of the name. This is an exception to the Little Endian order format for transmitting multi-byte parameters.

Command Parameters:

None.

Return Parameters:**Status:***Size: 1 Byte*

Value	Parameter Description
0x00	Read_Local_Name command succeeded
0x01-0xFF	Read_Local_Name command failed see Table 6.1 on page 748 for list of Error Codes

Name:*Size: 248 Bytes*

Value	Parameter Description
	A UTF-8 encoded User Friendly Descriptive Name for the device. The UTF-8 encoded Name can be up to 248 bytes in length. If it is shorter than 248 bytes, the end is indicated by a NULL byte (0x00).

Event(s) generated (unless masked away):

When the Read_Local_Name command has completed a Command Complete event will be generated.

4.7.13 Read_Conn ction_Accept_Timeout

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Connection_Accept_Timeout	0x0015		Status, Conn_Accept_Timeout

Description:

This command will read the value for the Connection_Accept_Timeout configuration parameter. The Connection_Accept_Timeout configuration parameter allows the Bluetooth hardware to automatically deny a connection request after a specified time period has occurred and the new connection is not accepted. The parameter defines the time duration from when the Host Controller sends a Connection Request event until the Host Controller will automatically reject an incoming connection.

Command Parameters:

None.

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_Connection_Accept_Timeout command succeeded.
0x01-0xFF	Read_Connection_Accept_Timeout command failed. See Table 6.1 on page 745 for list of Error Codes.

Conn_Accept_Timeout:

Size: 2 Bytes

Value	Parameter Description
N = 0xFFFF	Connection Accept Timeout measured in Number of Baseband slots. Interval Length = $N * 0.625 \text{ msec}$ (1 Baseband slot) Range for N: 0x0001 – 0xB540 Time Range: 0.625 msec -29 seconds

Event(s) generated (unless masked away):

When the Read_Connection_Timeout command has completed, a Command Complete event will be generated.

4.7.14 Write_Connection_Accept_Timeout

Command	OCF	Command Parameters	Return Parameters
HCI_Write_Connection_Accept_Timeout	0x0016	Conn_Accept_Timeout	Status

Description:

This command will write the value for the Connection_Accept_Timeout configuration parameter. The Connection_Accept_Timeout configuration parameter allows the Bluetooth hardware to automatically deny a connection request after a specified time interval has occurred and the new connection is not accepted. The parameter defines the time duration from when the Host Controller sends a Connection Request event until the Host Controller will automatically reject an incoming connection.

Command Parameters:*Conn_Accept_Timeout:**Size: 2 Bytes*

Value	Parameter Description
N = 0xXXXX	Connection Accept Timeout measured in Number of Baseband slots. Interval Length = $N * 0.625 \text{ msec}$ (1 Baseband slot) Range for N: 0x0001 – 0xB540 Time Range: 0.625 msec - 29 seconds Default: N = 0x1FA0 Time = 5 Sec

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Write_Connection_Accept_Timeout command succeeded.
0x01-0xFF	Write_Connection_Accept_Timeout command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Connection_Accept_Timeout command has completed, a Command Complete event will be generated.

4.7.15 Read_Page_Timeout

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Page_Timeout	0x0017		Status, Page_Timeout

Description:

This command will read the value for the Page_Timeout configuration parameter. The Page_Timeout configuration parameter defines the maximum time the local Link Manager will wait for a baseband page response from the remote device at a locally initiated connection attempt. If this time expires and the remote device has not responded to the page at baseband level, the connection attempt will be considered to have failed.

Command Parameters:

None.

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_Page_Timeout command succeeded.
0x01-0xFF	Read_Page_Timeout command failed. See Table 6.1 on page 745 for list of Error Codes.

Page_Timeout:

Size: 2 Bytes

Value	Parameter Description
N = 0xXXXX	Page Timeout measured in Number of Baseband slots. Interval Length = $N * 0.625 \text{ msec}$ (1 Baseband slot) Range for N: 0x0001 – 0xFFFF Time Range: 0.625 msec -40.9 Seconds

Event(s) generated (unless masked away):

When the Read_Page_Timeout command has completed, a Command Complete event will be generated.

4.7.16 Write_Page_Timeout

Command	OCF	Command Parameters	Return Parameters
HCI_Write_Page_Timeout	0x0018	Page_Timeout	Status

Description:

This command will write the value for the Page_Timeout configuration parameter. The Page_Timeout configuration parameter defines the maximum time the local Link Manager will wait for a baseband page response from the remote device at a locally initiated connection attempt. If this time expires and the remote device has not responded to the page at baseband level, the connection attempt will be considered to have failed.

Command Parameters:*Page_Timeout:**Size: 2 Bytes*

Value	Parameter Description
0	Illegal Page Timeout. Must be larger than 0.
N = 0xXXXX	Page Timeout measured in Number of Baseband slots. Interval Length = $N * 0.625 \text{ msec}$ (1 Baseband slot) Range for N: 0x0001 – 0xFFFF Time Range: 0.625 msec - 40.9 Seconds Default: N = 0x2000 Time = 5.12 Sec

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Write_Page_Timeout command succeeded.
0x01-0xFF	Write_Page_Timeout command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Page_Timeout command has completed, a Command Complete event will be generated.

4.7.17 Read_Scan_Enable

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Scan_Enable	0x0019		Status, Scan_Enable

Description:

This command will read the value for the Scan_Enable parameter. The Scan_Enable parameter controls whether or not the Bluetooth device will periodically scan for page attempts and/or inquiry requests from other Bluetooth devices. If Page_Scan is enabled, then the device will enter page scan mode based on the value of the Page_Scan_Interval and Page_Scan_Window parameters. If Inquiry_Scan is enabled, then the device will enter Inquiry Scan mode based on the value of the Inquiry_Scan_Interval and Inquiry_Scan_Window parameters.

Command Parameters:

None.

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_Scan_Enable command succeeded.
0x01-0xFF	Read_Scan_Enable command failed. See Table 6.1 on page 745 for list of Error Codes.

Scan_Enable:

Size: 1 Byte

Value	Parameter Description
0x00	No Scans enabled.
0x01	Inquiry Scan enabled. Page Scan disabled.
0x02	Inquiry Scan disabled. Page Scan enabled.
0x03	Inquiry Scan enabled. Page Scan enabled.

Event(s) generated (unless masked away):

When the Read_Scan_Enable command has completed, a Command Complete event will be generated.

4.7.18 Write_Scan_Enable

Command	OCF	Command Parameters	Return Parameters
HCI_Write_Scan_Enable	0x001A	Scan_Enable	Status

Description:

This command will write the value for the Scan_Enable parameter. The Scan_Enable parameter controls whether or not the Bluetooth device will periodically scan for page attempts and/or inquiry requests from other Bluetooth devices. If Page_Scan is enabled, then the device will enter page scan mode based on the value of the Page_Scan_Interval and Page_Scan_Window parameters. If Inquiry_Scan is enabled, then the device will enter Inquiry Scan mode based on the value of the Inquiry_Scan_Interval and Inquiry_Scan_Window parameters.

Command Parameters:*Scan_Enable:**Size: 1 Byte*

Value	Parameter Description
0x00	No Scans enabled. Default.
0x01	Inquiry Scan enabled. Page Scan disabled.
0x02	Inquiry Scan disabled. Page Scan enabled.
0x03	Inquiry Scan enabled. Page Scan enabled.

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Write_Scan_Enable command succeeded.
0x01-0xFF	Write_Scan_Enable command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Scan_Enable command has completed, a Command Complete event will be generated.

4.7.19 Read_Page_Scan_Activity

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Page_Scan_Activity	0x001B		Status, Page_Scan_Interval, Page_Scan_Window

Description:

This command will read the value for Page_Scan_Activity configuration parameters. The Page_Scan_Interval configuration parameter defines the amount of time between consecutive page scans. This time interval is defined from when the Host Controller started its last page scan until it begins the next page scan. The Page_Scan_Window configuration parameter defines the amount of time for the duration of the page scan. The Page_Scan_Window can only be less than or equal to the Page_Scan_Interval.

Note: Page Scan is only performed when Page_Scan is enabled (see 4.7.17 and 4.7.18).

A changed Page_Scan_Interval could change the local Page_Scan_Repetition_Mode (see "Baseband Specification" on page 33, Keyword: SR-Mode).

Command Parameters:

None.

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_Page_Scan_Activity command succeeded.
0x01-0xFF	Read_Page_Scan_Activity command failed. See Table 6.1 on page 745 for list of Error Codes.

Page_Scan_Interval:

Size: 2 Bytes

Value	Parameter Description
N = 0xXXXX	Size: 2 Bytes Range: 0x0012 – 0x1000 Time = N * 0.625 msec Range: 11.25 msec – 2560 msec

*Page_Scan_Window:**Size: 2 Bytes*

Value	Parameter Description
N = 0xFFFF	Size: 2 Bytes Range: 0x0012 – 0x1000 Time = N * 0.625 msec Range: 11.25 msec – 2560 msec

Event(s) generated (unless masked away):

When the Read_Page_Scan_Activity command has completed, a Command Complete event will be generated.

4.7.20 Write_Page_Scan_Activity

Command	OCF	Command Parameters	Return Parameters
HCI_Write_Page_Scan_Activity	0x001C	Page_Scan_Interval, Page_Scan_Window	Status

Description:

This command will write the value for Page_Scan_Activity configuration parameter. The Page_Scan_Interval configuration parameter defines the amount of time between consecutive page scans. This is defined as the time interval from when the Host Controller started its last page scan until it begins the next page scan. The Page_Scan_Window configuration parameter defines the amount of time for the duration of the page scan. The Page_Scan_Window can only be less than or equal to the Page_Scan_Interval.

Note: Page Scan is only performed when Page_Scan is enabled (see 4.7.17 and 4.7.18). A changed Page_Scan_Interval could change the local Page_Scan_Repetition_Mode (see "Baseband Specification" on page 33, Keyword: SR-Mode).

Command Parameters:*Page_Scan_Interval:**Size: 2 Bytes*

Value	Parameter Description
N = 0xXXXX	Size: 2 Bytes Range: 0x0012 – 0x1000 Time = N * 0.625 msec Range: 11.25 msec – 2560 msec Default: N = 0x0800 Time = 1.28 Sec

*Page_Scan_Window:**Size: 2 Bytes*

Value	Parameter Description
N = 0xXXXX	Size: 2 Bytes Range: 0x0012 – 0x1000 Time = N * 0.625 msec Range: 11.25 msec – 2560 msec Default: N = 0x0012 Time = 11.25 msec

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Write_Page_Scan_Activity command succeeded.
0x01-0xFF	Write_Page_Scan_Activity command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Page_Scan_Activity command has completed, a Command Complete event will be generated.

4.7.21 Read_Inquiry_Scan_Activity

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Inquiry_Scan_Activity	0x001D		Status, Inquiry_Scan_Interval, Inquiry_Scan_Window

Description:

This command will read the value for Inquiry_Scan_Activity configuration parameter. The Inquiry_Scan_Interval configuration parameter defines the amount of time between consecutive inquiry scans. This is defined as the time interval from when the Host Controller started its last inquiry scan until it begins the next inquiry scan.

The Inquiry_Scan_Window configuration parameter defines the amount of time for the duration of the inquiry scan. The Inquiry_Scan_Window can only be less than or equal to the Inquiry_Scan_Interval.

Note: Inquiry Scan is only performed when Inquiry_Scan is enabled see 4.7.17 and 4.7.18).

Command Parameters:

None.

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_Inquiry_Scan_Activity command succeeded.
0x01-0xFF	Read_Inquiry_Scan_Activity command failed. See Table 6.1 on page 745 for list of Error Codes.

Inquiry_Scan_Interval:

Size: 2 Bytes

Value	Parameter Description
N = 0xXXXX	Size: 2 Bytes Range: 0x0012 – 0x1000 Time = N * 0.625 msec Range: 11.25 – 2560 msec

*Inquiry_Scan_Window:**Size: 2 Bytes*

Value	Parameter Description
N = 0xFFFF	Size: 2 Bytes Range: 0x0012 – 0x1000 Time = N * 0.625 msec Range: 0.625 msec – 2560 msec

Event(s) generated (unless masked away):

When the Read_Inquiry_Scan_Activity command has completed, a Command Complete event will be generated.

4.7.22 Write_Inquiry_Scan_Activity

Command	OCF	Command Parameters	Return Parameters
HCI_Write_Inquiry_Scan_Activity	0x001E	Inquiry_Scan_Interval, Inquiry_Scan_Window	Status

Description:

This command will write the value for Inquiry_Scan_Activity configuration parameter. The Inquiry_Scan_Interval configuration parameter defines the amount of time between consecutive inquiry scans. This is defined as the time interval from when the Host Controller started its last inquiry scan until it begins the next inquiry scan.

The Inquiry_Scan_Window configuration parameter defines the amount of time for the duration of the inquiry scan. The Inquiry_Scan_Window can only be less than or equal to the Inquiry_Scan_Interval.

Note: Inquiry Scan is only performed when Inquiry_Scan is enabled (see 4.7.17 and 4.7.18).

Command Parameters:*Inquiry_Scan_Interval:**Size: 2 Bytes*

Value	Parameter Description
N = 0xXXXX	Size: 2 Bytes Range: 0x0012 – 0x1000 Time = N * 0.625 msec Range: 11.25 – 2560 msec Default: N = 0x0800 Time = 1.28 Sec

*Inquiry_Scan_Window:**Size: 2 Bytes*

Value	Parameter Description
N = 0xXXXX	Size: 2 Bytes Range: 0x0012 – 0x1000 Time = N * 0.625 msec Range: 11.25 msec – 2560 msec Default: N = 0x0012 Time = 11.25 msec

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Write_Inquiry_Scan_Activity command succeeded.
0x01-0xFF	Write_Inquiry_Scan_Activity command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Inquiry_Scan_Activity command has completed, a Command Complete event will be generated.

4.7.23 Read_Authentication_Enable

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Authentication_Enable	0x001F		Status, Authentication_Enable

Description:

This command will read the value for the Authentication_Enable parameter. The Authentication_Enable parameter controls if the local device requires to authenticate the remote device at connection setup (between the Create_Connection command or acceptance of an incoming ACL connection and the corresponding Connection Complete event). At connection setup, only the device(s) with the Authentication_Enable parameter enabled will try to authenticate the other device.

Note: Changing this parameter does not affect existing connections.

Command Parameters:

None.

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_Authentication_Enable command succeeded.
0x01-0xFF	Read_Authentication_Enable command failed. See Table 6.1 on page 745 for list of Error Codes.

Authentication_Enable:

Size: 1 Byte

Value	Parameter Description
0x00	Authentication disabled.
0x01	Authentication enabled for all connections.
0x02-0xFF	Reserved

Event(s) generated (unless masked away):

When the Read_Authentication_Enable command has completed, a Command Complete event will be generated.

4.7.24 Write_Authentication_Enable

Command	OCF	Command Parameters	Return Parameters
HCI_Write_Authentication_Enable	0x0020	Authentication_Enable	Status

Description:

This command will write the value for the Authentication_Enable parameter. The Authentication_Enable parameter controls if the local device requires to authenticate the remote device at connection setup (between the Create_Connection command or acceptance of an incoming ACL connection and the corresponding Connection Complete event). At connection setup, only the device(s) with the Authentication_Enable parameter enabled will try to authenticate the other device.

Note: Changing this parameter does not affect existing connections.

Command Parameters:*Authentication_Enable:**Size: 1 Byte*

Value	Parameter Description
0x00	Authentication disabled. Default.
0x01	Authentication enabled for all connection.
0x02-0xFF	Reserved

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Write Authentication_Enable command succeeded.
0x01-0xFF	Write Authentication_Enable command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Authentication_Enable command has completed, a Command Complete event will be generated.

4.7.25 Read_Encryption_Mode

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Encryption_Mode	0x0021		Status, Encryption_Mode

Description:

This command will read the value for the Encryption_Mode parameter. The Encryption_Mode parameter controls if the local device requires encryption to the remote device at connection setup (between the Create_Connection command or acceptance of an incoming ACL connection and the corresponding Connection Complete event). At connection setup, only the device(s) with the Authentication_Enable parameter enabled and Encryption_Mode parameter enabled will try to encrypt the connection to the other device.

Note: Changing this parameter does not affect existing connections.

Command Parameters:

None.

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_Encryption_Mode command succeeded.
0x01-0xFF	Read_Encryption_Mode command failed. See Table 6.1 on page 745 for list of Error Codes.

Encryption_Mode:

Size: 1 Byte

Value	Parameter Description
0x00	Encryption disabled.
0x01	Encryption only for point-to-point packets.
0x02	Encryption for both point-to-point and broadcast packets.
0x03-0xFF	Reserved.

Event(s) generated (unless masked away):

When the Read_Encryption_Mode command has completed, a Command Complete event will be generated.

4.7.26 Writ _Encryption_Mode

Command	OCF	Command Parameters	Return Parameters
HCI_Write_Encryption_Mode	0x0022	Encryption_Mode	Status

Description:

This command will write the value for the Encryption_Mode parameter. The Encryption_Mode parameter controls if the local device requires encryption to the remote device at connection setup (between the Create_Connection command or acceptance of an incoming ACL connection and the corresponding Connection Complete event). At connection setup, only the device(s) with the Authentication_Enable parameter enabled and Encryption_Mode parameter enabled will try to encrypt the connection to the other device.

Note: Changing this parameter does not affect existing connections.

A temporary link key must be used when both broadcast and point-to-point traffic shall be encrypted.

The Host must not specify the Encryption_Mode parameter with more encryption capability than its local device currently supports, although the parameter is used to request the encryption capability to the remote device. Note that the Host must not request the command with the Encryption_Mode parameter set to either 0x01 or 0x02, when the local device does not support encryption. Also note that the Host must not request the command with the parameter set to 0x02, when the local device does not support broadcast encryption.

Note that the actual Encryption_Mode to be returned in an event for a new connection (or in a Connection Complete event) will only support a part of the capability, when the local device requests more encryption capability than the current remote device supports. For example, 0x00 will always be returned in the event when the remote device supports no encryption, and either 0x00 or 0x01 will be returned when it supports only point-to-point encryption.

Command Parameters:

Encryption_Mode:

Size: 1 Byte

Value	Parameter Description
0x00	Encryption disabled. Default.
0x01	Encryption only for point-to-point packets.
0x02	Encryption for both point-to-point and broadcast packets.
0x03-0xFF	Reserved.

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Write_Encryption_Mode command succeeded.
0x01-0xFF	Write_Encryption_Mode command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Encryption_Mode command has completed, a Command Complete event will be generated.

4.7.27 Read_Class_of_Device

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Class_of_Device	0x0023		Status, Class_of_Device

Description:

This command will read the value for the Class_of_Device parameter. The Class_of_Device parameter is used to indicate the capabilities of the local device to other devices.

Command Parameters:

None.

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_Class_of_Device command succeeded.
0x01-0xFF	Read_Class_of_Device command failed. See Table 6.1 on page 745 for list of Error Codes.

Class_of_Device:

Size: 3 Bytes

Value	Parameter Description
0xXXXXXX	Class of Device for the device.

Event(s) generated (unless masked away):

When the Read_Class_of_Device command has completed, a Command Complete event will be generated.

4.7.28 Writ _Class_of_Device

Command	OCF	Command Parameters	Return Parameters
HCI_Write_Class_of_Device	0x0024	Class_of_Device	Status

Description:

This command will write the value for the Class_of_Device parameter. The Class_of_Device parameter is used to indicate the capabilities of the local device to other devices.

Command Parameters:*Class_of_Device:**Size: 3 Bytes*

Value	Parameter Description
0xXXXXXX	Class of Device for the device.

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Write_Class_of_Device command succeeded.
0x01-0xFF	Write_Class_of_Device command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Class_of_Device command has completed, a Command Complete event will be generated.

4.7.29 Read_Voice_Setting

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Voice_Setting	0x0025		Status, Voice_Setting

Description:

This command will read the values for the Voice_Setting parameter. The Voice_Setting parameter controls all the various settings for voice connections. These settings apply to all voice connections, and **cannot** be set for individual voice connections. The Voice_Setting parameter controls the configuration for voice connections: Input Coding, Air coding format, input data format, Input sample size, and linear PCM parameter.

Command Parameters:

None.

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_Voice_Setting command succeeded.
0x01-0xFF	Read_Voice_Setting command failed. See Table 6.1 on page 745 for list of Error Codes.

Voice_Setting:

Size: 2 Bytes (10 Bits meaningful)

Value	Parameter Description
00XXXXXXXX	Input Coding: Linear
01XXXXXXXX	Input Coding: μ -law Input Coding
10XXXXXXXX	Input Coding: A-law Input Coding
11XXXXXXXX	Reserved for Future Use
XX00XXXXXX	Input Data Format: 1's complement
XX01XXXXXX	Input Data Format: 2's complement
XX10XXXXXX	Input Data Format: Sign-Magnitude
XX11XXXXXX	Reserved for Future Use
XXXX0XXXXX	Input Sample Size: 8-bit (only for Liner PCM)
XXXX1XXXXX	Input Sample Size: 16-bit (only for Liner PCM)

Value	Parameter Description
XXXXXnnnXX	Linear_PCM_Bit_Pos: # bit positions that MSB of sample is away from starting at MSB (only for Liner PCM).
XXXXXXXXXX00	Air Coding Format: CVSD
XXXXXXXXXX01	Air Coding Format: μ -law
XXXXXXXXXX10	Air Coding Format: A-law
XXXXXXXXXX11	Reserved

Event(s) generated (unless masked away):

When the Read_Voice_Setting command has completed, a Command Complete event will be generated.

4.7.30 Writ_Voic_Setting

Command	OCF	Command Parameters	Return Parameters
HCI_Write_Voice_Setting	0x0026	Voice_Setting	Status

Description:

This command will write the values for the Voice_Setting parameter. The Voice_Setting parameter controls all the various settings for the voice connections. These settings apply to all voice connections and **cannot** be set for individual voice connections. The Voice_Setting parameter controls the configuration for voice connections: Input Coding, Air coding format, input data format, Input sample size, and linear PCM parameter.

Command Parameters:*Voice_Setting:**Size: 2 Bytes (10 Bits meaningful)*

Value	Parameter Description
00XXXXXXXX	Input Coding: Linear
01XXXXXXXX	Input Coding: μ -law Input Coding
10XXXXXXXX	Input Coding: A-law Input Coding
11XXXXXXXX	Reserved for Future Use
XX00XXXXXX	Input Data Format: 1's complement
XX01XXXXXX	Input Data Format: 2's complement
XX10XXXXXX	Input Data Format: Sign-Magnitude
XX11XXXXXX	Reserved for Future Use
XXXX0XXXXX	Input Sample Size: 8 bit (only for Liner PCM)
XXXX1XXXXX	Input Sample Size: 16 bit (only for Liner PCM)
XXXXXnnnXX	Linear_PCM_Bit_Pos: # bit positions that MSB of sample is away from starting at MSB (only for Liner PCM)
XXXXXXXX00	Air Coding Format: CVSD
XXXXXXXX01	Air Coding Format: μ -law
XXXXXXXX10	Air Coding Format: A-law
XXXXXXXX11	Reserved
0001100000	Default Condition

R turn Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Write_Voice_Setting command succeeded.
0x01-0xFF	Write_Voice_Setting command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Voice_Setting command has completed, a Command Complete event will be generated.

4.7.31 Read_Automatic_Flush_Timeout

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Automatic_Flush_Timeout	0x0027	Connection_Handle	Status, Connection_Handle, Flush_Timeout

Description:

This command will read the value for the Flush_Timeout parameter for the specified connection handle. The Flush_Timeout parameter is used for ACL connections ONLY. The Flush_Timeout parameter defines the amount of time before all chunks of the L2CAP packet, of which a baseband packet is currently being transmitted, are automatically flushed by the Host Controller. The timeout period starts when a transmission attempt is made for the first baseband packet of an L2CAP packet. This allows ACL packets to be automatically flushed without the Host device issuing a Flush command. The Read_Automatic_Flush_Timeout command provides support for isochronous data, such as video. When the L2CAP packet that is currently being transmitted is automatically 'flushed', the Failed Contact Counter is incremented by one. The first chunk of the next L2CAP packet to be transmitted for the specified connection handle may already be stored in the Host Controller. In that case, the transmission of the first baseband packet containing data from that L2CAP packet can begin immediately. If the next L2CAP packet is not stored in the Host Controller, all data that is sent to the Host Controller after the flush for the same connection handle will be discarded by the Host Controller until an HCI Data Packet having the start Packet_Boundary_Flag (0x02) is received. When this happens, a new transmission attempt will be made.

Command Parameters:*Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xXXXX	Specifies which Connection Handle's Flush Timeout to read. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Read_Automatic_Flush_Timeout command succeeded.
0x01-0xFF	Read_Automatic_Flush_Timeout command failed. See Table 6.1 on page 745 for list of Error Codes.

Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)**

Value	Parameter Description
0xFFFF	Specifies which Connection Handle's Flush Timeout has been read. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Flush_Timeout:**Size: 2 Bytes**

Value	Parameter Description
0	Timeout = ∞ ; No Automatic Flush
N = 0xFFFF	Flush Timeout = $N * 0.625 \text{ msec}$ Size: 11 bits Range: 0x0001 – 0x07FF

Event(s) generated (unless masked away):

When the Read_Automatic_Flush_Timeout command has completed, a Command Complete event will be generated.

4.7.32 Write_Automatic_Flush_Timeout

Command	OCF	Command Parameters	Return Parameters
HCI_Write_Automatic_Flush_Timeout	0x0028	Connection_Handle, Flush_Timeout	Status, Connection_Handle

Description:

This command will write the value for the Flush_Timeout parameter for the specified connection handle. The Flush_Timeout parameter is used for ACL connections ONLY. The Flush_Timeout parameter defines the amount of time before all chunks of the L2CAP packet, of which a baseband packet is currently being transmitted, are automatically flushed by the Host Controller. The timeout period starts when a transmission attempt is made for the first baseband packet of an L2CAP packet. This allows ACL packets to be automatically flushed without the Host device issuing a Flush command. The Write_Automatic_Flush_Timeout command provides support for isochronous data, such as video. When the L2CAP packet that is currently being transmitted is automatically 'flushed', the Failed Contact Counter is incremented by one. The first chunk of the next L2CAP packet to be transmitted for the specified connection handle may already be stored in the Host Controller. In that case, the transmission of the first baseband packet containing data from that L2CAP packet can begin immediately. If the next L2CAP packet is not stored in the Host Controller, all data that is sent to the Host Controller after the flush for the same connection handle will be discarded by the Host Controller until an HCI Data Packet having the start Packet_Boundary_Flag (0x02) is received. When this happens, a new transmission attempt will be made.

Command Parameters:*Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xXXXX	Specifies which Connection Handle's Flush Timeout to write to. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

*Flush_Timeout:**Size: 2 Bytes*

Value	Parameter Description
0	Timeout = ∞; No Automatic Flush. Default.
N = 0xXXXX	Flush Timeout = N * 0.625 msec Size: 11 bits Range: 0x0001 – 0x07FF

Return Parameters:**Status:****Size: 1 Byte**

Value	Parameter Description
0x00	Write_Automatic_Flush_Timeout command succeeded.
0x01-0xFF	Write_Automatic_Flush_Timeout command failed. See Table 6.1 on page 745 for list of Error Codes.

Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)**

Value	Parameter Description
0xFFFF	Specifies which Connection Handle's Flush Timeout has been written. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Event(s) generated (unless masked away):

When the Write_Automatic_Flush_Timeout command has completed, a Command Complete event will be generated.

4.7.33 Read_Num_Broadcast_R transmissions

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Num_Broadcast_Retransmissions	0x0029		Status, Num_Broadcast_Retran

Description:

This command will read the device's parameter value for the Number of Broadcast Retransmissions. Broadcast packets are not acknowledged and are unreliable. The Number of Broadcast Retransmissions parameter is used to increase the reliability of a broadcast message by retransmitting the broadcast message multiple times. This parameter defines the number of times the device will retransmit a broadcast data packet. This parameter should be adjusted as the link quality measurement changes.

Command Parameters:

None.

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_Num_Broadcast_Retransmissions command succeeded.
0x01-0xFF	Read_Num_Broadcast_Retransmissions command failed. See Table 6.1 on page 745 for list of Error Codes.

Num_Broadcast_Retran:

Size: 1 Byte

Value	Parameter Description
N = 0xFF	Number of Broadcast Retransmissions = N Range 0x00-0xFF

Event(s) generated (unless masked away):

When the Read_Num_Broadcast_Retransmission command has completed, a Command Complete event will be generated.

4.7.34 Write_Num_Broadcast_Retransmissions

Command	OCF	Command Parameters	Return Parameters
HCI_Write_Num_Broadcast_Retransmissions	0x002A	Num_Broadcast_Retran	Status

Description:

This command will write the device's parameter value for the Number of Broadcast Retransmissions. Broadcast packets are not acknowledged and are unreliable. The Number of Broadcast Retransmissions parameter is used to increase the reliability of a broadcast message by retransmitting the broadcast message multiple times. This parameter defines the number of times the device will retransmit a broadcast data packet. This parameter should be adjusted as link quality measurement change.

Command Parameters:*Num_Broadcast_Retran:**Size: 1 Byte*

Value	Parameter Description
N = 0xXX	Number of Broadcast Retransmissions = N Range 0x00-0xFF Default: N = 0x01

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Write_Num_Broadcast_Retransmissions command succeeded.
0x01-0xFF	Write_Num_Broadcast_Retransmissions command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Num_Broadcast_Retransmissions command has completed, a Command Complete event will be generated.

4.7.35 R ad_Hold_Mode_Activity

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Hold_Mode_Activity	0x002B		Status, Hold_Mode_Activity

Description:

This command will read the value for the Hold_Mode_Activity parameter. The Hold_Mode_Activity value is used to determine what activities should be suspended when the device is in hold mode. After the hold period has expired, the device will return to the previous mode of operation. Multiple hold mode activities may be specified for the Hold_Mode_Activity parameter by performing a bitwise OR operation of the different activity types. If no activities are suspended, then all of the current Periodic Inquiry, Inquiry Scan, and Page Scan settings remain valid during the Hold Mode. If the Hold_Mode_Activity parameter is set to Suspend Page Scan, Suspend Inquiry Scan, and Suspend Periodic Inquiries, then the device can enter a low-power state during the Hold Mode period, and all activities are suspended. Suspending multiple activities can be specified for the Hold_Mode_Activity parameter by performing a bitwise OR operation of the different activity types. The Hold Mode Activity is only valid if all connections are in Hold Mode.

Command Parameters:

None.

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Read_Hold_Mode_Activity command succeeded.
0x01-0xFF	Read_Hold_Mode_Activity command failed. See Table 6.1 on page 745 for list of Error Codes.

*Hold_Mode_Activity:**Size: 1 Byte*

Value	Parameter Description
0x00	Maintain current Power State.
0x01	Suspend Page Scan.
0x02	Suspend Inquiry Scan.
0x04	Suspend Periodic Inquiries.
0x08-0xFF	R served for Future Use.

Event(s) generated (unless masked away):

When the Read_Hold_Mode_Activity command has completed, a Command Complete event will be generated.

4.7.36 Write_Hold_Mode_Activity

Command	OCF	Command Parameters	Return Parameters
HCI_Write_Hold_Mode_Activity	0x002C	Hold_Mode_Activity	Status

Description:

This command will write the value for the Hold_Mode_Activity parameter. The Hold_Mode_Activity value is used to determine what activities should be suspended when the device is in hold mode. After the hold period has expired, the device will return to the previous mode of operation. Multiple hold mode activities may be specified for the Hold_Mode_Activity parameter by performing a bitwise OR operation of the different activity types. If no activities are suspended, then all of the current Periodic Inquiry, Inquiry Scan, and Page Scan settings remain valid during the Hold Mode. If the Hold_Mode_Activity parameter is set to Suspend Page Scan, Suspend Inquiry Scan, and Suspend Periodic Inquiries, then the device can enter a low power state during the Hold Mode period and all activities are suspended. Suspending multiple activities can be specified for the Hold_Mode_Activity parameter by performing a bitwise OR operation of the different activity types. The Hold Mode Activity is only valid if all connections are in Hold Mode.

Command Parameters:*Hold_Mode_Activity:**Size: 1 Byte*

Value	Parameter Description
0x00	Maintain current Power State. Default.
0x01	Suspend Page Scan.
0x02	Suspend Inquiry Scan.
0x04	Suspend Periodic Inquiries.
0x08-0xFF	Reserved for Future Use.

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Write_Hold_Mode_Activity command succeeded.
0x01-0xFF	Write_Hold_Mode_Activity command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Hold_Mode_Activity command has completed, a Command Complete event will be generated.

4.7.37 Read_Transmit_Power_Level

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Transmit_Power_Level	0x002D	Connection_Handle, Type	Status, Connection_Handle, Transmit_Power_Level

Description:

This command will read the values for the Transmit_Power_Level parameter for the specified Connection_Handle. The Connection_Handle must be a Connection_Handle for an ACL connection.

Command Parameters:*Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xFFFF	Specifies which Connection_Handle's Transmit Power Level setting to read. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

*Type:**Size: 1 Byte*

Value	Parameter Description
0x00	Read Current Transmit Power Level.
0x01	Read Maximum Transmit Power Level.
0x02-0xFF	Reserved

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Read_Transmit_Power_Level command succeeded.
0x01-0xFF	Read_Transmit_Power_Level command failed. See Table 6.1 on page 745 for list of Error Codes.

*Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xFFFF	Specifies which Connection_Handle's Transmit Power Level setting is returned. Range : 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

*Transmit_Power_Level:**Size: 1 Byte*

Value	Parameter Description
N = 0xXX	Size: 1 Byte (signed integer) Range: $-30 \leq N \leq 20$ Units: dBm

Event(s) generated (unless masked away):

When the Read_Transmit_Power_Level command has completed, a Command Complete event will be generated.

4.7.38 Read_SCO_Flow_Control_Enable

Command	OCF	Command Parameters	Return Parameters
HCI_Read_SCO_Flow_Control_Enable	0x002E		Status, SCO_Flow_Control_Enable

Description:

The Read_SCO_Flow_Control_Enable command provides the ability to read the SCO_Flow_Control_Enable setting. By using this setting, the Host can decide if the Host Controller will send Number Of Completed Packets events for SCO Connection Handles. This setting allows the Host to enable and disable SCO flow control.

Note: the SCO_Flow_Control_Enable setting can only be changed if no connections exist.

Command Parameters:

None.

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_SCO_Flow_Control_Enable command succeeded
0x01-0xFF	Read_SCO_Flow_Control_Enable command failed see Table 6.1 on page 746 for list of Error Codes

SCO_Flow_Control_Enable:

Size: 1 Byte

Value	Parameter Description
0x00	SCO Flow Control is disabled. No Number of Completed Packets events will be sent from the Host Controller for SCO Connection Handles.
0x01	SCO Flow Control is enabled. Number of Completed Packets events will be sent from the Host Controller for SCO Connection Handles.

Event(s) generated (unless masked away):

When the Read_SCO_Flow_Control_Enable command has completed a Command Complete event will be generated.

4.7.39 Write_SCO_Flow_Control_Enable

Command	OCF	Command Parameters	Return Parameters
HCI_Write_SCO_Flow_Control_Enable	0x002F	SCO_Flow_Control_Enable	Status

Description:

The Write_SCO_Flow_Control_Enable command provides the ability to write the SCO_Flow_Control_Enable setting. By using this setting, the Host can decide if the Host Controller will send Number Of Completed Packets events for SCO Connection Handles. This setting allows the Host to enable and disable SCO flow control.

Note: the SCO_Flow_Control_Enable setting can only be changed if no connections exist.

Command Parameters:*SCO_Flow_Control_Enable:**Size: 1 Byte*

Value	Parameter Description
0x00	SCO Flow Control is disabled. No Number of Completed Packets events will be sent from the Host Controller for SCO Connection Handles. Default
0x01	SCO Flow Control is enabled. Number of Completed Packets events will be sent from the Host Controller for SCO Connection Handles.

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Write_SCO_Flow_Control_Enable command succeeded
0x01-0xFF	Write_SCO_Flow_Control_Enable command failed see Table 6.1 on page 746 for list of Error Codes

Event(s) generated (unless masked away):

When the Write_SCO_Flow_Control_Enable command has completed a Command Complete event will be generated.

4.7.40 Set_Host_Controller_To_Host_Flow_Control

Command	OCF	Command Parameters	Return Parameters
HCI_Set_Host_Controller_To_Host_Flow_Control	0x0031	Flow_Control_Enable	Status

Description:

This command is used by the Host to turn flow control on or off in the direction from the Host Controller to the Host. If flow control is turned off, the Host should not send the Host_Number_Of_Completed_Packets command. That command will be ignored by the Host Controller if it is sent by the Host and flow control is off.

Command Parameters:*Flow_Control_Enable:**Size: 1 Byte*

Value	Parameter Description
0x00	Flow control off in direction from Host Controller to Host. Default.
0x01	Flow control on in direction from Host Controller to Host.
0x02-0xFF	Reserved

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Set_Host_Controller_To_Host_Flow_Control command succeeded.
0x01-0xFF	Set_Host_Controller_To_Host_Flow_Control command failed. See Table 6.1 on page 7450 for list of Error Codes.

Event(s) generated (unless masked away):

When the Set_Host_Controller_To_Host_Flow_Control command has completed, a Command Complete event will be generated.

4.7.41 Host_Buffer_Size

Command	OCF	Command Parameters	Return Parameters
HCI_Host_Buffer_Size	0x0033	Host_ACL_Data_Packet_Length, Host_SCO_Data_Packet_Length, Host_Total_Num_ACL_Data_Packets, Host_Total_Num_SCO_Data_Packets	Status

Description:

The Host_Buffer_Size command is used by the Host to notify the Host Controller about the maximum size of the data portion of HCI ACL and SCO Data Packets sent from the Host Controller to the Host. The Host Controller will segment the data to be transmitted from the Host Controller to the Host according to these sizes, so that the HCI Data Packets will contain data with up to these sizes. The Host_Buffer_Size command also notifies the Host Controller about the total number of HCI ACL and SCO Data Packets that can be stored in the data buffers of the Host. If flow control from the Host Controller to the Host is turned off, and the Host_Buffer_Size command has not been issued by the Host, this means that the Host Controller will send HCI Data Packets to the Host with any lengths the Host Controller wants to use, and it is assumed that the data buffer sizes of the Host are unlimited. If flow control from the Host controller to the Host is turned on, the Host_Buffer_Size command must after a power-on or a reset always be sent by the Host before the first Host_Number_Of_Completed_Packets command is sent.

(The Set_Host_Controller_To_Host_Flow_Control command is used to turn flow control on or off.) The Host_ACL_Data_Packet_Length command parameter will be used to determine the size of the L2CAP segments contained in ACL Data Packets, which are transferred from the Host Controller to the Host. The Host_SCO_Data_Packet_Length command parameter is used to determine the maximum size of HCI SCO Data Packets. Both the Host and the Host Controller must support command and event packets, where the data portion (excluding header) contained in the packets is 255 bytes in size.

The Host_Total_Num_ACL_Data_Packets command parameter contains the total number of HCI ACL Data Packets that can be stored in the data buffers of the Host. The Host Controller will determine how the buffers are to be divided between different Connection Handles. The Host_Total_Num_SCO_Data_Packets command parameter gives the same information for HCI SCO Data Packets.

Note: the Host_ACL_Data_Packet_Length and Host_SCO_Data_Packet_Length command parameters do not include the length of the HCI Data Packet header.

Command Parameters:*Host_ACL_Data_Packet_Length:**Size: 2 Bytes*

Value	Parameter Description
0xFFFF	Maximum length (in bytes) of the data portion of each HCI ACL Data Packet that the Host is able to accept.

*Host_SCO_Data_Packet_Length:**Size: 1 Byte*

Value	Parameter Description
0xFF	Maximum length (in bytes) of the data portion of each HCI SCO Data Packet that the Host is able to accept.

*Host_Total_Num_ACL_Data_Packets:**Size: 2 Bytes*

Value	Parameter Description
0xFFFF	Total number of HCI ACL Data Packets that can be stored in the data buffers of the Host.

*Host_Total_Num_SCO_Data_Packets:**Size: 2 Bytes*

Value	Parameter Description
0xFFFF	Total number of HCI SCO Data Packets that can be stored in the data buffers of the Host.

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Host_Buffer_Size command succeeded.
0x01-0xFF	Host_Buffer_Size command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Host_Buffer_Size command has completed, a Command Complete event will be generated.

4.7.42 Host_Number_Of_Completed_Packets

Command	OCF	Command Parameters	Return Parameters
HCI_Host_Number_Of_Completed_Packets	0x0035	Number_Of_Handles, Connection_Handle[], Host_Num_Of_Completed_Packets [i]	

Description:

The Host_Number_Of_Completed_Packets command is used by the Host to indicate to the Host Controller the number of HCI Data Packets that have been completed for each Connection Handle since the previous Host_Number_Of_Completed_Packets command was sent to the Host Controller. This means that the corresponding buffer space has been freed in the Host. Based on this information, and the Host_Total_Num_ACL_Data_Packets and Host_Total_Num_SCO_Data_Packets command parameters of the Host_Buffer_Size command, the Host Controller can determine for which Connection Handles the following HCI Data Packets should be sent to the Host. The command should only be issued by the Host if flow control in the direction from the Host Controller to the Host is on and there is at least one connection, or if the Host Controller is in local loopback mode. Otherwise, the command will be ignored by the Host Controller. While the Host has HCI Data Packets in its buffers, it must keep sending the Host_Number_Of_Completed_Packets command to the Host Controller at least periodically, until it finally reports that all buffer space in the Host used by ACL Data Packets has been freed. The rate with which this command is sent is manufacturer specific.

(The Set_Host_Controller_To_Host_Flow_Control command is used to turn flow control on or off.) If flow control from the Host controller to the Host is turned on, the Host_Buffer_Size command must after a power-on or a reset always be sent by the Host before the first Host_Number_Of_Completed_Packets command is sent.

Note: the Host_Number_Of_Completed_Packets command is a special command in the sense that no event is normally generated after the command has completed. The command may be sent at any time by the Host when there is at least one connection, or if the Host Controller is in local loopback mode independent of other commands. The normal flow control for commands is not used for the Host_Number_Of_Completed_Packets command.

Command Parameters:*Number_Of_Handles:**Size: 1 Byte*

Value	Parameter Description
0xXX	The number of Connection Handles and Host_Num_Of_Completed_Packets parameters pairs contained in this command. Range: 0-255

*Connection_Handle[i]: Size: Number_Of_Handles*2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xFFFF	Connection Handle Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

*Host_Num_Of_Completed_Packets [i]: Size: Number_Of_Handles * 2 Bytes*

Value	Parameter Description
N = 0xFFFF	The number of HCI Data Packets that have been completed for the associated Connection Handle since the previous time the event was returned. Range for N: 0x0000-0xFFFF

Return Parameters:

None.

Event(s) generated (unless masked away):

Normally, no event is generated after the Host_Number_Of_Completed_Packets command has completed. However, if the Host_Number_Of_Completed_Packets command contains one or more invalid parameters, the Host Controller will return a Command Complete event with a failure status indicating the Invalid HCI Command Parameters error code. The Host may send the Host_Number_Of_Completed_Packets command at any time when there is at least one connection, or if the Host Controller is in local loopback mode. The normal flow control for commands is not used for this command.

4.7.43 Read_Link_Supervision_Timeout

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Link_Supervision_Timeout	0x0036	Connection_Handle	Status, Connection_Handle, Link_Supervision_Timeout

Description:

This command will read the value for the Link_Supervision_Timeout parameter for the device. The Link_Supervision_Timeout parameter is used by the master or slave Bluetooth device to monitor link loss. If, for any reason, no Baseband packets are received from that Connection_Handle for a duration longer than the Link_Supervision_Timeout, the connection is disconnected. The same timeout value is used for both SCO and ACL connections for the device specified by the Connection_Handle.

Note: the Connection_Handle used for this command must be the ACL connection to the appropriate device. This command will set the Link_Supervision_Timeout values for other SCO Connection_Handle to that device.

Note: Setting the Link_Supervision_Timeout to No Link_Supervision_Timeout (0x0000) will disable the Link_Supervision_Timeout check for the specified Connection_Handle. This makes it unnecessary for the master of the piconet to unpark and then park each Bluetooth Device every ~40 seconds. By using the No Link_Supervision_Timeout setting, the scalability of the Park mode is not limited.

Command Parameters:*Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xFFFF	Specifies which Connection_Handle's Link Supervision Timeout value is to be read. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Read_Link_Supervision_Timeout command succeeded.
0x01-0xFF	Read_Link_Supervision_Timeout command failed. See Table 6.1 on page 745 for list of Error Codes.

Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)**

Value	Parameter Description
0xFFFF	Specifies which Connection Handle's Link Supervision Timeout value was read. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Link_Supervision_Timeout:**Size: 2 Bytes**

Value	Parameter Description
0x0000	No Link_Supervision_Timeout.
N = 0xFFFF	Measured in Number of Baseband slots $\text{Link_Supervision_Timeout} = N * 0.625 \text{ msec (1 Baseband slot)}$ Range for N: 0x0001 – 0xFFFF Time Range: 0.625ms - 40.9 sec

Event(s) generated (unless masked away):

When the Read_Link_Supervision_Timeout command has completed, a Command Complete event will be generated.

4.7.44 Write Link_Supervisi n_Timeout

Command	OCF	Command Parameters	Return Parameters
HCI_Write_Link_Supervision_Timeout	0x0037	Connection_Handle, Link_Supervision_Timeout	Status, Connection_Handle

Description:

This command will write the value for the Link_Supervision_Timeout parameter for the device. The Link_Supervision_Timeout parameter is used by the master or slave Bluetooth device to monitor link loss. If, for any reason, no Baseband packets are received from that Connection_Handle for a duration longer than the Link_Supervision_Timeout, the connection is disconnected. The same timeout value is used for both SCO and ACL connections for the device specified by the Connection_Handle.

Note: the Connection_Handle used for this command must be the ACL connection to the appropriate device. This command will set the Link_Supervision_Timeout values for other SCO Connection_Handle to that device.

Note: Setting the Link_Supervision_Timeout parameter to No Link_Supervision_Timeout (0x0000) will disable the Link_Supervision_Timeout check for the specified Connection Handle. This makes it unnecessary for the master of the piconet to unpark and then park each Bluetooth Device every ~40 seconds. By using the No Link_Supervision_Timeout setting, the scalability of the Park mode is not limited.

Command Parameters:*Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xFFFF	Specifies which Connection Handle's Link Supervision Timeout value is to be written. Range: 0x0000-0x0EFF (0x0F00 - 0xFFFF Reserved for future use)

*Link_Supervision_Timeout:**Size: 2 Bytes*

Value	Parameter Description
0x0000	No Link_Supervision_Timeout.
N = 0xFFFF	Measured in Number of Baseband slots Link_Supervision_Timeout = N*0.625 msec (1 Baseband slot) Range for N: 0x0001 – 0xFFFF Time Range: 0.625ms – 40.9 sec Default: N = 0x7D00 Link_Supervision_Timeout = 20 sec

Return Parameters:**Status:****Size: 1 Byte**

Value	Parameter Description
0x00	Write_Link_Supervision_Timeout command succeeded.
0x01-0xFF	Write_Link_Supervision_Timeout command failed. See Table 6.1 on page 745 for list of Error Codes.

Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)**

Value	Parameter Description
0xFFFF	Specifies which Connection Handle's Link Supervision Timeout value was written. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Event(s) generated (unless masked away):

When the Write_Link_Supervision_Timeout command has completed, a Command Complete event will be generated.

4.7.45 Read_Number_Of_Supported_IAC

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Number_Of_Supported_IAC	0x0038		Status, Num_Support_IAC

Description:

This command will read the value for the number of Inquiry Access Codes (IAC) that the local Bluetooth device can simultaneously listen for during an Inquiry Scan. All Bluetooth devices are required to support at least one IAC, the General Inquiry Access Code (GIAC or UIAC), but some Bluetooth devices support additional IACs.

Command Parameters:

None

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_Number_Of_Supported_IAC command succeeded.
0x01-0xFF	Read_Number_Of_Supported_IAC command failed. See Table 6.1 on page 745 for list of Error Codes.

Num_Support_IAC

Size: 1 Byte

Value	Parameter Description
0xXX	Specifies the number of Supported IAC that the local Bluetooth device can simultaneously listen for during an Inquiry Scan. Range: 0x01-0x40

Event(s) generated (unless masked away):

When the Read_Number_Of_Supported_IAC command has completed, a Command Complete event will be generated.

4.7.46 Read_Current_IAC_LAP

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Current_IAC_LAP	0x0039		Status, Num_Current_IAC, IAC_LAP[i]

Description:

This command reads the LAP(s) used to create the Inquiry Access Codes (IAC) that the local Bluetooth device is simultaneously scanning for during Inquiry Scans. All Bluetooth devices are required to support at least one IAC (GIAC or UIAC). Some Bluetooth devices support additional IACs.

Command Parameters:

None

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_Current_IAC_LAP command succeeded.
0x01-0xFF	Read_Current_IAC_LAP command failed. See Table 6.1 on page 745 for list of Error Codes.

Num_Current_IAC

Size: 1 Byte

Value	Parameter Description
0xXX	Specifies the number of IACs which are currently in use by the local Bluetooth device to simultaneously listen for during an Inquiry Scan. Range: 0x01-0x40

IAC_LAP[i]

*Size: 3 Bytes * Num_Current_IAC*

Value	Parameter Description
0XXXXXX	LAPs used to create the IAC which is currently in use by the local Bluetooth device to simultaneously listen for during an Inquiry Scan. Range: 0x9E8B00-0x9E8B3F

Event(s) generated (unless masked away):

When the Read_Current_IAC_LAP command has completed, a Command Complete event will be generated.

4.7.47 Write_Current_IAC_LAP

Command	OCF	Command Parameters	Return Parameters
HCI_Write_Current_IAC_LAP	0x003A	Num_Current_IAC, IAC_LAP[i]	Status

Description:

This command writes the LAP(s) used to create the Inquiry Access Codes (IAC) that the local Bluetooth device is simultaneously scanning for during Inquiry Scans. All Bluetooth devices are required to support at least one IAC (GIAC or UIAC). Some Bluetooth devices support additional IACs. Therefore, the LAP used to create the GIAC or UIAC must be among the IAC_LAP parameters of this command.

Note: this command writes over the current IACs used by the Bluetooth device. If the value of the NumCurrentIAC is more than the number of supported IACs, then only the first, X Inquiry Access Codes (where X equals the number of supported IACs) will be stored without any error.

Command Parameters:*Num_Current_IAC**Size: 1 Byte*

Value	Parameter Description
0xXX	Specifies the number of IACs which are currently in use by the local Bluetooth device to simultaneously listen for during an Inquiry Scan. Range: 0x01-0x40

*IAC_LAP[i]**Size: 3 Bytes * Num_Current_IAC*

Value	Parameter Description
0xXXXXXX	LAP(s) used to create IAC which is currently in use by the local Bluetooth device to simultaneously listen for during an Inquiry Scan. Range: 0x9E8B00-0x9E8B3F. The GIAC is the default IAC to be used. If additional IACs are supported, additional default IAC will be determined by the manufacturer.

Return Parameters:**Status:****Size: 1 Byte**

Value	Parameter Description
0x00	Write_Current_IAC_LAP command succeeded.
0x01-0xFF	Write_Current_IAC_LAP command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Current_IAC_LAP command has completed, a Command Complete event will be generated.

4.7.48 Read_Page_Scan_Period_Mode

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Page_Scan_Period_Mode	0x003B		Status, Page_Scan_Period_Mode

Description:

This command is used to read the mandatory Page_Scan_Period_Mode of the local Bluetooth device. Every time an inquiry response message is sent, the Bluetooth device will start a timer (T_mandatory_pscan), the value of which is dependent on the Page_Scan_Period_Mode. As long as this timer has not expired, the Bluetooth device will use the Page_Scan_Period_Mode for all following page scans.

Note: the timer T_mandatory_pscan will be reset at each new inquiry response. For details see the "Baseband Specification" on page 33. (Keyword: SP-Mode, FHS-Packet, T_mandatory_pscan, Inquiry-Response).

After transmitting one or more inquiry response (FHS) packets as a result of the inquiry scan process, the local Bluetooth device is allowed to enter the page scan state using mandatory page scan mode regardless of the setting of the Scan_Enable parameter.

Command Parameters:

None

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_Page_Scan_Period_Mode command succeeded.
0x01-0xFF	Read_Page_Scan_Period_Mode command failed. See Table 6.1 on page 745 for list of Error Codes.

Page_Scan_Period_Mode:

Size: 1 Byte

Value	Parameter Description
0x00	P0
0x01	P1
0x02	P2
0x03-0xFF	Reserved.

Event(s) generated (unless masked away):

When the Read_Page_Scan_Period_Mode command has completed, a Command Complete event will be generated.

4.7.49 Write_Page_Scan_Period_Mode

Command	OCF	Command Parameters	Return Parameters
HCI_Write_Page_Scan_Period_Mode	0x003C	Page_Scan_Period_Mode	Status

Description:

This command is used to write the mandatory Page_Scan_Period_Mode of the local Bluetooth device. Every time an inquiry response message is sent, the Bluetooth device will start a timer (T_mandatory_pscan), the value of which is dependent on the Page_Scan_Period_Mode. As long as this timer has not expired, the Bluetooth device will use the Page_Scan_Period_Mode for all following page scans.

Note: the timer T_mandatory_pscan will be reset at each new inquiry response. For details see the "Baseband Specification" on page 33. (Keyword: SP-Mode, FHS-Packet, T_mandatory_pscan, Inquiry-Response).

After transmitting one or more inquiry response (FHS) packets as a result of the inquiry scan process, the local Bluetooth device is allowed to enter the page scan state using mandatory page scan mode regardless of the setting of the Scan_Enable parameter.

Command Parameters:*Page_Scan_Period_Mode:**Size: 1 Byte*

Value	Parameter Description
0x00	P0. Default.
0x01	P1
0x02	P2
0x03-0xFF	Reserved.

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Write_Page_Scan_Period_Mode command succeeded.
0x01-0xFF	Write_Page_Scan_Period_Mode command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Page_Scan_Period_Mode command has completed, a Command Complete event will be generated.

4.7.50 Read_Page_Scan_Mode

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Page_Scan_Mode	0x003D		Status, Page_Scan_Mode

Description:

This command is used to read the default page scan mode of the local Bluetooth device. The Page_Scan_Mode parameter indicates the page scan mode that is used for default page scan. Currently one mandatory page scan mode and three optional page scan modes are defined. Following an inquiry response, if the Baseband timer T_mandatory_pscan has not expired, the mandatory page scan mode must be applied. For details see the "Baseband Specification" on page 33 (Keyword: Page-Scan-Mode, FHS-Packet, T_mandatory_pscan)

Command Parameters:

None

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_Page_Scan_Mode command succeeded.
0x01-0xFF	Read_Page_Scan_Mode command failed. See Table 6.1 on page 745 for list of Error Codes.

Page_Scan_Mode:

Size: 1 Byte

Value	Parameter Description
0x00	Mandatory Page Scan Mode
0x01	Optional Page Scan Mode I
0x02	Optional Page Scan Mode II
0x03	Optional Page Scan Mode III
0x04-0xFF	Reserved

Event(s) generated (unless masked away):

When the Read_Page_Scan_Mode command has completed, a Command Complete event will be generated.

4.7.51 Writ _Pag _Scan_Mod

Command	OCF	Command Parameters	Return Parameters
HCI_Write_Page_Scan_Mode	0x003E	Page_Scan_Mode	Status

Description:

This command is used to write the default page scan mode of the local Bluetooth device. The Page_Scan_Mode parameter indicates the page scan mode that is used for the default page scan. Currently, one mandatory page scan mode and three optional page scan modes are defined. Following an inquiry response, if the Baseband timer T_mandatory_pscan has not expired, the mandatory page scan mode must be applied. For details see the "Baseband Specification" on page 33. (Keyword: Page-Scan-Mode, FHS-Packet, T_mandatory_pscan).

Command Parameters:*Page_Scan_Mode:**Size: 1 Byte*

Value	Parameter Description
0x00	Mandatory Page Scan Mode. Default.
0x01	Optional Page Scan Mode I
0x02	Optional Page Scan Mode II
0x03	Optional Page Scan Mode III
0x04-0xFF	Reserved.

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Write_Page_Scan_Mode command succeeded.
0x01-0xFF	Write_Page_Scan_Mode command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Page_Scan_Mode command has completed, a Command Complete event will be generated.

4.8 INFORMATIONAL PARAMETERS

The Informational Parameters are fixed by the manufacturer of the Bluetooth hardware. These parameters provide information about the Bluetooth device and the capabilities of the Host Controller, Link Manager, and Baseband. The host device cannot modify any of these parameters. For Informational Parameters Commands, the OGF is defined as 0x04

Command	Command Summary Description
Read_Local_Version_Information	The Read_Local_Version_Information command will read the values for the version information for the local Bluetooth device.
Read_Local_Supported_Features	The Read_Local_Supported_Features command requests a list of the supported features for the local device.
Read_Buffer_Size	The Read_Buffer_Size command returns the size of the HCI buffers. These buffers are used by the Host Controller to buffer data that is to be transmitted.
Read_Country_Code	The Read_Country_Code command will read the value for the Country Code status parameter. The Country Code defines which range of frequency band of the ISM 2.4 GHz band will be used by the device.
Read_BD_ADDR	The Read_BD_ADDR command will read the value for the BD_ADDR parameter. The BD_ADDR is a 48-bit unique identifier for a Bluetooth device.

4.8.1 Read_Local_Version_Information

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Local_Version_Information	0x0001		Status, HCI Version, HCI Revision, LMP Version, Manufacturer_Name, LMP Subversion

Description:

This command will read the values for the version information for the local Bluetooth device. The version information consists of two parameters: the version and revision parameters.

The version parameter defines the major hardware version of the Bluetooth hardware. The version parameter only changes when new versions of the Bluetooth hardware are produced for new Bluetooth SIG specifications. The version parameter is controlled by the SIG.

The revision parameter should be controlled by the manufacturer and should be changed as needed. The Manufacturer_Name parameter indicates the manufacturer of the local Bluetooth module as specified by the LMP definition of this parameter. The subversion parameter should be controlled by the manufacturer and should be changed as needed. The subversion parameter defines the various revisions that each version of the Bluetooth hardware will go through as design processes change and errors are fixed. This allows the software to determine what Bluetooth hardware is being used, and to work around various bugs in the hardware if necessary.

Command Parameters:

None.

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_Local_Version_Information command succeeded.
0x01-0xFF	Read_Local_Version_Information command failed. See Table 6.1 on page 745 for list of Error Codes.

HCI_Version:**Size: 1 Byte**

Value	Parameter Description
0xXX	Version of the Current HCI in the Bluetooth hardware . 0x00: Bluetooth HCI Specification 1.0 0x01-0xFF: Reserved

HCI_Revision:**Size: 2 Bytes**

Value	Parameter Description
0XXXXX	Revision of the Current HCI in the Bluetooth hardware.

LMP_Version:**Size: 1 Byte**

Value	Parameter Description
0xXX	Version of the Current LMP in the Bluetooth Hardware, see Table 5.2 on page 231 in the Link Manager Protocol for assigned values (VersNr).

Manufacturer_Name:**Size: 2 Bytes**

Value	Parameter Description
0XXXXX	Manufacturer Name of the Bluetooth Hardware, see Table 5.2 on page 231 in the Link Manager Protocol for assigned values (Compld).

LMP_Subversion:**Size: 2 Bytes**

Value	Parameter Description
0XXXXX	Subversion of the Current LMP in the Bluetooth Hardware, see Table 5.2 on page 231 in the Link Manager Protocol for assigned values (SubVersNr).

Event(s) generated (unless masked away):

When the Read_Local_Version_Information command has completed, a Command Complete event will be generated.

4.8.2 Read_Local_Supported_Features

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Local_Supported_Features	0x0003		Status, LMP_Features

Description:

This command requests a list of the supported features for the local device. This command will return a list of the LMP features. For details see "Link Manager Protocol" on page 185.

Command Parameters:

None.

Return Parameters:

Status:

Size: 1 Byte.

Value	Parameter Description
0x00	Read_Local_Supported_Features command succeeded.
0x01-0xFF	Read_Local_Supported_Features command failed. See Table 6.1 on page 745 for list of Error Codes.

LMP_Features:

Size: 8 Bytes

Value	Parameter Description
0XXXXXXXXX XXXXXXXXXX	Bit Mask List of LMP features. For details see "Link Manager Protocol" on page 185

Event(s) generated (unless masked away):

When the Read_Local_Supported_Features command has completed, a Command Complete event will be generated.

4.8.3 Read_Buffer_Size

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Buffer_Size	0x0005		Status, HC_ACL_Data_Packet_Length, HC_SCO_Data_Packet_Length, HC_ Total_Num_ACL_Data_Packets, HC_Total_Num_SCO_Data_Packets

Description:

The Read_Buffer_Size command is used to read the maximum size of the data portion of HCI ACL and SCO Data Packets sent from the Host to the Host Controller. The Host will segment the data to be transmitted from the Host to the Host Controller according to these sizes, so that the HCI Data Packets will contain data with up to these sizes. The Read_Buffer_Size command also returns the total number of HCI ACL and SCO Data Packets that can be stored in the data buffers of the Host Controller. The Read_Buffer_Size command must be issued by the Host before it sends any data to the Host Controller.

The HC_ACL_Data_Packet_Length return parameter will be used to determine the size of the L2CAP segments contained in ACL Data Packets, which are transferred from the Host to the Host Controller to be broken up into baseband packets by the Link Manager. The HC_SCO_Data_Packet_Length return parameter is used to determine the maximum size of HCI SCO Data Packets. Both the Host and the Host Controller must support command and event packets, where the data portion (excluding header) contained in the packets is 255 bytes in size. The HC_Total_Num_ACL_Data_Packets return parameter contains the total number of HCI ACL Data Packets that can be stored in the data buffers of the Host Controller. The Host will determine how the buffers are to be divided between different Connection Handles. The

HC_Total_Num_SCO_Data_

Packets return parameter gives the same information but for HCI SCO Data Packets.

Note: the HC_ACL_Data_Packet_Length and HC_SCO_Data_Packet_Length return parameters do not include the length of the HCI Data Packet header.

Command Parameters:

None.

Return Parameters:**Status:****Size: 1 Byte**

Value	Parameter Description
0x00	Read_Buffer_Size command succeeded.
0x01-0xFF	Read_Buffer_Size command failed. See Table 6.1 on page 745 for list of Error Codes.

HC_ACL_Data_Packet_Length:**Size: 2 Bytes**

Value	Parameter Description
0xFFFF	Maximum length (in bytes) of the data portion of each HCI ACL Data Packet that the Host Controller is able to accept.

HC_SCO_Data_Packet_Length:**Size: 1 Byte**

Value	Parameter Description
0xFF	Maximum length (in bytes) of the data portion of each HCI SCO Data Packet that the Host Controller is able to accept.

HC_Total_Num_ACL_Data_Packets:**Size: 2 Bytes**

Value	Parameter Description
0xFFFF	Total number of HCI ACL Data Packets that can be stored in the data buffers of the Host Controller.

HC_Total_Num_SCO_Data_Packets:**Size: 2 Bytes**

Value	Parameter Description
0xFFFF	Total number of HCI SCO Data Packets that can be stored in the data buffers of the Host Controller.

Event(s) generated (unless masked away):

When the Read_Buffer_Size command has completed, a Command Complete event will be generated.

4.8.4 Read_Country_Code

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Country_Code	0x0007		Status, Country_Code

Description:

This command will read the value for the Country_Code return parameter. The Country_Code defines which range of frequency band of the ISM 2.4 GHz band will be used by the device. Each country has local regulatory bodies regulating which ISM 2.4 GHz frequency ranges can be used.

Command Parameters:

None.

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_Country_Code command succeeded.
0x01-0xFF	Read_Country_Code command failed. See Table 6.1 on page 745 for list of Error Codes.

Country_Code:

Size: 1 Byte

Value	Parameter Description
0x00	North America & Europe*
0x01	France
0x02	Spain
0x03	Japan
0x04-FF	Reserved for Future Use.

*. Except Spain and France

Event(s) generated (unless masked away):

When the Read_Country_Code command has completed, a Command Complete event will be generated.

4.8.5 Read_BD_ADDR

Command	OCF	Command Parameters	Return Parameters
HCI_Read_BD_ADDR	0x0009		Status, BD_ADDR

Description:

This command will read the value for the BD_ADDR parameter. The BD_ADDR is a 48-bit unique identifier for a Bluetooth device. See the "Baseband Specification" on page 33 for details of how BD_ADDR is used.

Command Parameters:

None.

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Read_BD_ADDR command succeeded.
0x01-0xFF	Read_BD_ADDR command failed. See Table 6.1 on page 745 for list of Error Codes.

BD_ADDR:

Size: 6 Bytes

Value	Parameter Description
0XXXXXXXXXXXXX	BD_ADDR of the Device

Event(s) generated (unless masked away):

When the Read_BD_ADDR command has completed, a Command Complete event will be generated.

4.9 STATUS PARAMETERS

The Host Controller modifies all status parameters. These parameters provide information about the current state of the Host Controller, Link Manager, and Baseband. The host device cannot modify any of these parameters other than to reset certain specific parameters. For the Status and baseband, the OGF is defined as 0x05

Command	Command Summary Description
Read_Failed_Contact_Counter	The Read_Failed_Contact_Counter will read the value for the Failed_Contact_Counter parameter for a particular connection to another device. The Failed_Contact_Counter records the number of consecutive incidents in which either the slave or master didn't respond after the flush timeout had expired, and the L2CAP packet that was currently being transmitted was automatically 'flushed'.
Reset_Failed_Contact_Counter	The Reset_Failed_Contact_Counter will reset the value for the Failed_Contact_Counter parameter for a particular connection to another device. The Failed_Contact_Counter records the number of consecutive incidents in which either the slave or master didn't respond after the flush timeout had expired and the L2CAP packet that was currently being transmitted was automatically 'flushed'.
Get_Link_Quality	The Get_Link_Quality command will read the value for the Link_Quality for the specified Connection Handle.
Read_RSSI	The Read_RSSI command will read the value for the Received Signal Strength Indication (RSSI) for a connection handle to another Bluetooth device.

4.9.1 Read_Failed_Contact_Count

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Failed_Contact_Counter	0x0001	Connection_Handle	Status, Connection_Handle, Failed_Contact_Counter

Description:

This command will read the value for the Failed_Contact_Counter parameter for a particular connection to another device. The Connection_Handle must be a Connection_Handle for an ACL connection. The Failed_Contact_Counter records the number of consecutive incidents in which either the slave or master didn't respond after the flush timeout had expired, and the L2CAP packet that was currently being transmitted was automatically 'flushed'. When this occurs, the Failed_Contact_Counter is incremented by 1. The Failed_Contact_Counter for a connection is reset to zero on the following conditions:

1. When a new connection is established
2. When the Failed_Contact_Counter is > zero and an L2CAP packet is acknowledged for that connection
3. When the Reset_Failed_Contact_Counter command has been issued

Command Parameters:*Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xFFFF	The Connection Handle for the Connection for which the Failed Contact Counter should be read. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Read_Failed_Contact_Counter command succeeded.
0x01-0xFF	Read_Failed_Contact_Counter command failed. See Table 6.1 on page 745 for list of Error Codes.

Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)**

Value	Parameter Description
0xFFFF	The Connection Handle for the Connection for which the Failed Contact Counter has been read. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Failed_Contact_Counter:**Size: 2 Bytes**

Value	Parameter Description
0xFFFF	Number of consecutive failed contacts for a connection corresponding to the connection handle.

Event(s) generated (unless masked away):

When the Read_Failed_Contact_Counter command has completed, a Command Complete event will be generated.

4.9.2 R s t_Failed_Contact_Counter

Command	OCF	Command Parameters	Return Parameters
HCI_Reset_Failed_Contact_Counter	0x0002	Connection_Handle	Status, Connection_Handle

Description:

This command will reset the value for the Failed_Contact_Counter parameter for a particular connection to another device. The Connection_Handle must be a Connection_Handle for an ACL connection. The Failed_Contact_Counter records the number of consecutive incidents in which either the slave or master didn't respond after the flush timeout had expired, and the L2CAP packet that was currently being transmitted was automatically 'flushed'. When this occurs, the Failed_Contact_Counter is incremented by 1. The Failed_Contact_Counter for a connection is reset to zero on the following conditions:

1. When a new connection is established
2. When the Failed_Contact_Counter is > zero and an L2CAP packet is acknowledged for that connection
3. When the Reset_Failed_Contact_Counter command has been issued

Command Parameters:*Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xXXXX	The Connection Handle for the Connection for which the Failed Contact Counter should be reset. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Reset_Failed_Contact_Counter command succeeded.
0x01-0xFF	Reset_Failed_Contact_Counter command failed. See Table 6.1 on page 745 for list of Error Codes.

Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)**

Value	Parameter Description
0xFFFF	The Connection Handle for the Connection for which the Failed Contact Counter has been reset. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Event(s) generated (unless masked away):

When the Reset_Failed_Contact_Counter command has completed, a Command Complete event will be generated.

4.9.3 G t_Link_Quality

Command	OCF	Command Parameters	Return Parameters
HCI_Get_Link_Quality	0x0003	Connection_Handle	Status, Connection_Handle, Link_Quality

Description:

This command will return the value for the Link_Quality for the specified Connection_Handle. The Connection_Handle must be a Connection_Handle for an ACL connection. This command will return a Link_Quality value from 0-255, which represents the quality of the link between two Bluetooth devices. The higher the value, the better the link quality is. Each Bluetooth module vendor will determine how to measure the link quality.

Command Parameters:

Connection_Handle:

Size: 2 Bytes (12 Bits meaningful)

Value	Parameter Description
0xFFFF	The Connection_Handle for the connection for which link quality parameters are to be read. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Get_Link_Quality command succeeded.
0x01-0xFF	Get_Link_Quality command failed. See Table 6.1 on page 745 for list of Error Codes.

Connection_Handle:

Size: 2 Bytes (12 Bits meaningful)

Value	Parameter Description
0xFFFF	The Connection_Handle for the connection for which the link quality parameter has been read. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Link_Quality:**Size: 1 Byte**

Value	Parameter Description
0xXX	The current quality of the Link connection between the local device and the remote device specified by the Connection Handle Range: 0x00 – 0xFF The higher the value, the better the link quality is.

Event(s) generated (unless masked away):

When the Get_Link_Quality command has completed, a Command Complete event will be generated.

4.9.4 R ad_RSSI

Command	OCF	Command Parameters	Return Parameters
HCI_Read_RSSI	0x0005	Connection_Handle	Status, Connection_Handle,RSSI

Description:

This command will read the value for the difference between the measured Received Signal Strength Indication (RSSI) and the limits of the Golden Receive Power Range (see Radio Specification Section 4.7 on page 26) for a connection handle to another Bluetooth device. The Connection_Handle must be a Connection_Handle for an ACL connection. Any positive RSSI value returned by the Host Controller indicates how many dB the RSSI is above the upper limit, any negative value indicates how many dB the RSSI is below the lower limit. The value zero indicates that the RSSI is inside the Golden Receive Power Range.

Note: how accurate the dB values will be depends on the Bluetooth hardware. The only requirements for the hardware are that the Bluetooth device is able to tell whether the RSSI is inside, above or below the Golden Device Power Range.

Command Parameters:*Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xFFFF	The Connection Handle for the Connection for which the RSSI is to be read. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Read_RSSI command succeeded.
0x01-0xFF	Read_RSSI command failed. See Table 6.1 on page 745 for list of Error Codes.

*Connection_Handle:**Size: 2 Bytes (12 Bits meaningful)*

Value	Parameter Description
0xFFFF	The Connection Handle for the Connection for which the RSSI has been read. Range: 0x0000-0x0EFF (0x0F00 - 0x0FFF Reserved for future use)

RSSI:

Size: 1 Byte

Value	Parameter Description
N = 0xXX	Size: 1 Byte (signed integer) Range: $-128 \leq N \leq 127$ Units: dB

Event(s) generated (unless masked away):

When the Read_RSSI command has completed, a Command Complete event will be generated.

4.10 TESTING COMMANDS

The Testing commands are used to provide the ability to test various functionalities of the Bluetooth hardware. These commands provide the ability to arrange various conditions for testing. For the Testing Commands, the OGF is defined as 0x06

Command	Command Summary Description
Read_Loopback_Mode	The Read_Loopback_Mode will read the value for the setting of the Host Controllers Loopback Mode. The setting of the Loopback Mode will determine the path of information.
Write_Loopback_Mode	The Write_Loopback_Mode will write the value for the setting of the Host Controllers Loopback Mode. The setting of the Loopback Mode will determine the path of information.
Enable_Device_Under_Test_Mode	The Enable_Device_Under_Test_Mode command will allow the local Bluetooth module to enter test mode via LMP test commands. The Host issues this command when it wants the local device to be the DUT for the Testing scenarios as described in the Bluetooth Test Mode document.

4.10.1 Read_Loopback_Mode

Command	OCF	Command Parameters	Return Parameters
HCI_Read_Loopback_Mode	0x0001		Status, Loopback_Mode

Description:

This command will read the value for the setting of the Host Controller's Loopback Mode. The setting of the Loopback Mode will determine the path of information. In Non-testing Mode operation, the Loopback Mode is set to Non-testing Mode and the path of the information is as specified by the Bluetooth specifications. In Local Loopback Mode, every Data Packet (ACL and SCO) and Command Packet that is sent from the Host to the Host Controller is sent back with no modifications by the Host Controller, as shown in Fig. 4.5 on page 697.

When the Bluetooth Host Controller enters Local Loopback Mode, it shall respond with four Connection Complete events, one for an ACL channel and three for SCO channels, so that the Host gets connection handles to use when sending ACL and SCO data. When in Local Loopback Mode the Host Controller loops back commands and data to the Host. The Loopback Command event is used to loop back commands that the Host sends to the Host Controller.

There are some commands that are not looped back in Local Loopback Mode: Reset, Set_Host_Controller_To_Host_Flow_Control, Host_Buffer_Size, Host_Number_Of_Completed_Packets, Read_Buffer_Size, Read_Loopback_Mode and Write_Loopback_Mode. These commands should be executed in the way they are normally executed. The commands Reset and Write_Loopback_Mode can be used to exit local loopback mode. If Write_Loopback_Mode is used to exit Local Loopback Mode, four Disconnection Complete events should be sent to the Host, corresponding to the Connection Complete events that were sent when entering Local Loopback Mode. Furthermore, no connections are allowed in Local Loopback mode. If there is a connection and there is an attempt to set the device to Local Loopback Mode, the attempt will be refused. When the device is in Local Loopback Mode, the Host Controller will refuse incoming connection attempts. This allows the Host Controller Transport Layer to be tested without any other variables.

If a device is set to Remote Loopback Mode, it will send back all data (ACL and SCO) that comes over the air, and it will only allow a maximum of one ACL connection and three SCO connections – and these should be all to the same remote device. If there already are connections to more than one remote device and there is an attempt to set the local device to Remote Loopback Mode, the attempt will be refused. See Fig. 4.6 on page 697 where the rightmost device is set to Remote Loopback Mode and the leftmost device is set to

Non-testing Mode. This allows the Bluetooth Air link to be tested without any other variables.

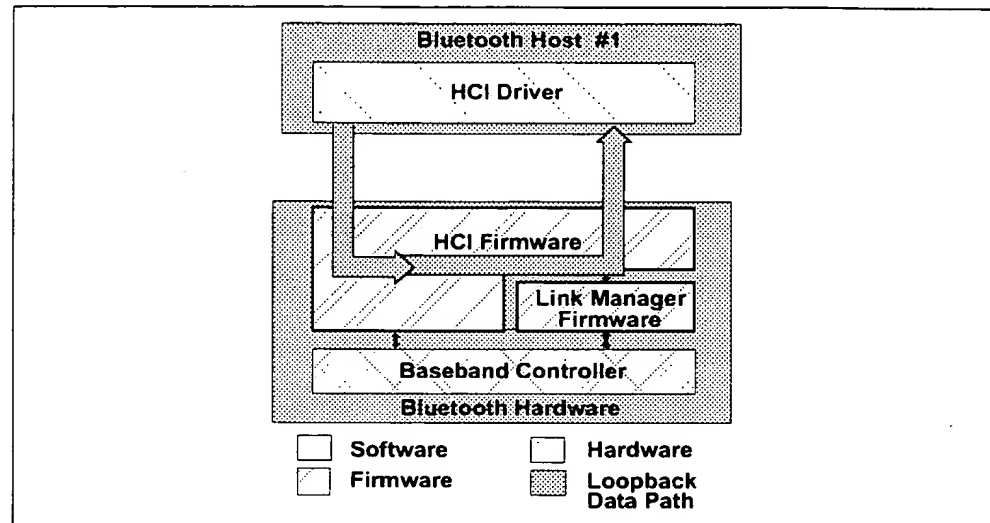


Figure 4.5: Local Loopback Mode

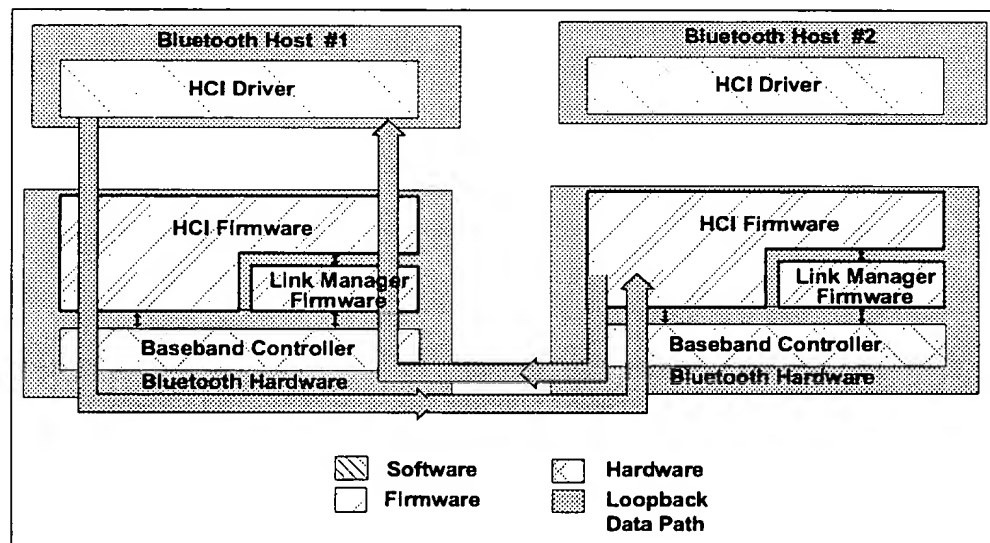


Figure 4.6: Remote Loopback Mode

Command Parameters:

None.

Return Parameters:*Status:**Size: 1 Byte*

Value	Parameter Description
0x00	Read_Loopback_Mode command succeeded.
0x01-0xFF	Read_Loopback_Mode command failed. See Table 2 on page 260 for list of Error Codes.

*Loopback_Mode:**Size: 1 Byte*

Value	Parameter Description
0x00	No Loopback mode enabled. Default.
0x01	Enable Local Loopback.
0x02	Enable Remote Loopback.
0x03-0xFF	Reserved for Future Use.

Event(s) generated (unless masked away):

When the Read_Loopback_Mode command has completed, a Command Complete event will be generated.

4.10.2 Write_Loopback_Mode

Command	OCF	Command Parameters	Return Parameters
HCI_Write_Loopback_Mode	0x0002	Loopback_Mode	Status

Description:

This command will write the value for the setting of the Host Controller's Loopback Mode. The setting of the Loopback Mode will determine the path of information. In Non-testing Mode operation, the Loopback Mode is set to Non-testing Mode and the path of the information as specified by the Bluetooth specifications. In Local Loopback Mode, every Data Packet (ACL and SCO) and Command Packet that is sent from the Host to the Host Controller is sent back with no modifications by the Host Controller, as shown in Fig. 4.7 on page 700.

When the Bluetooth Host Controller enters Local Loopback Mode, it shall respond with four Connection Complete events, one for an ACL channel and three for SCO channels, so that the Host gets connection handles to use when sending ACL and SCO data. When in Local Loopback Mode, the Host Controller loops back commands and data to the Host. The Loopback Command event is used to loop back commands that the Host sends to the Host Controller.

There are some commands that are not looped back in Local Loopback Mode: Reset, Set_Host_Controller_To_Host_Flow_Control, Host_Buffer_Size, Host_Number_Of_Completed_Packets, Read_Buffer_Size, Read_Loopback_Mode and Write_Loopback_Mode. These commands should be executed in the way they are normally executed. The commands Reset and Write_Loopback_Mode can be used to exit local loopback mode.

If Write_Loopback_Mode is used to exit Local Loopback Mode, four Disconnection Complete events should be sent to the Host corresponding to the Connection Complete events that were sent when entering Local Loopback Mode. Furthermore, no connections are allowed in Local Loopback mode. If there is a connection, and there is an attempt to set the device to Local Loopback Mode, the attempt will be refused. When the device is in Local Loopback Mode, the Host Controller will refuse incoming connection attempts. This allows the Host Controller Transport Layer to be tested without any other variables.

If a device is set to Remote Loopback Mode, it will send back all data (ACL and SCO) that comes over the air. It will only allow a maximum of one ACL connection and three SCO connections, and these should all be to the same remote device. If there already are connections to more than one remote device and there is an attempt to set the local device to Remote Loopback Mode, the attempt will be refused.

See Fig. 4.8 on page 700, where the rightmost device is set to Remote Loopback Mode and the leftmost device is set to Non-testing Mode. This allows the Bluetooth Air link to be tested without any other variables.

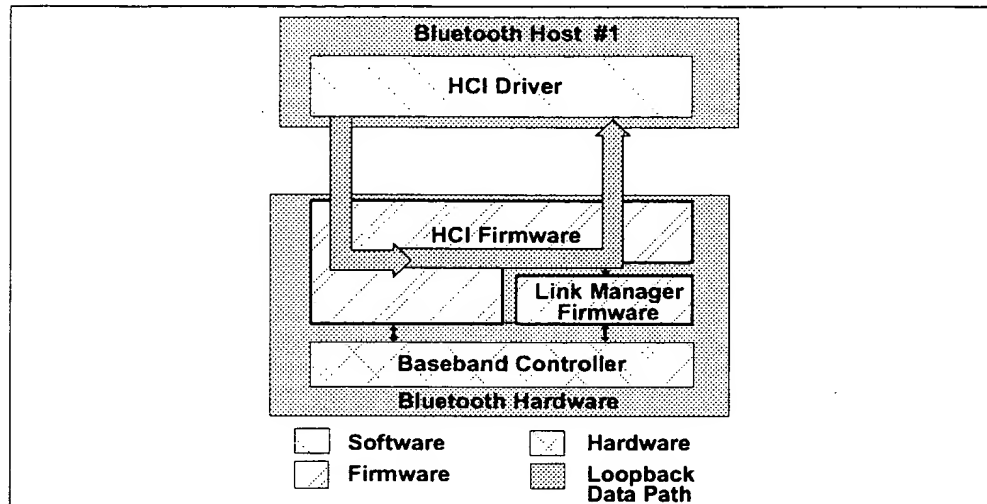


Figure 4.7: Local Loopback Mode

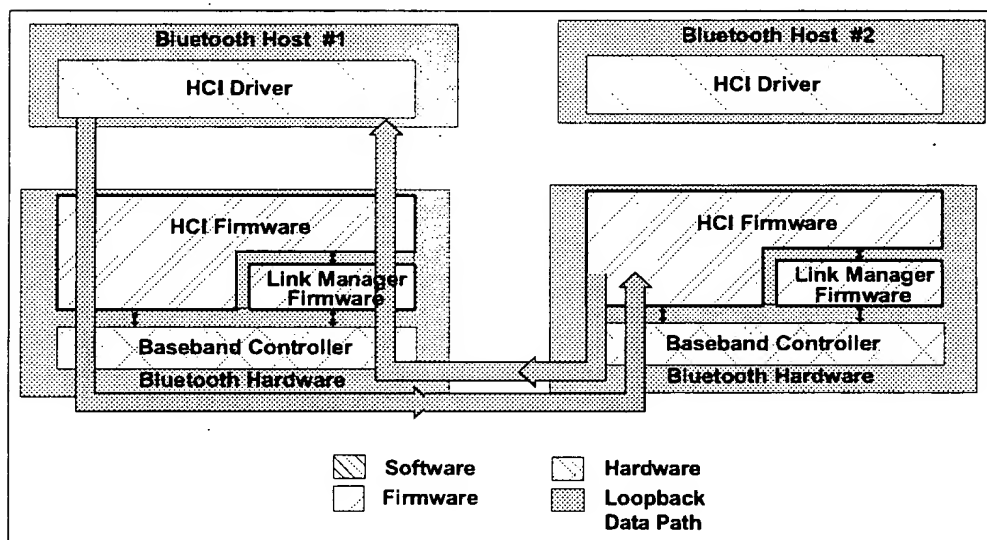


Figure 4.8: Remote Loopback Mode

Command Parameters:**Loopback_Mode:****Size: 1 Byte**

Value	Parameter Description
0x00	No Loopback mode enabled. Default.
0x01	Enable Local Loopback.
0x02	Enable Remote Loopback.
0x03-0xFF	Reserved for Future Use.

Return Parameters:**Status:****Size: 1 Byte**

Value	Parameter Description
0x00	Write_Loopback_Mode command succeeded.
0x01-0xFF	Write_Loopback_Mode command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Write_Loopback_Mode command has completed, a Command Complete event will be generated.

4.10.3 Enable_Device_Under_Test_Mode

Command	OCF	Command Parameters	Return Parameters
HCI_Enable_Device_Under_Test_Mode	0x0003		Status

Description:

The Enable_Device_Under_Test_Mode command will allow the local Bluetooth module to enter test mode via LMP test commands. For details see "Link Manager Protocol" on page 185. The Host issues this command when it wants the local device to be the DUT for the Testing scenarios as described in the "Bluetooth Test Mode" on page 803. When the Host Controller receives this command, it will complete the command with a Command Complete event. The Host Controller functions as normal until the remote tester issues the LMP test command to place the local device into Device Under Test mode. To disable and exit the Device Under Test Mode, the Host can issue the HCI_Reset command. This command prevents remote Bluetooth devices from causing the local Bluetooth device to enter test mode without first issuing this command.

Command Parameters:

None.

Return Parameters:

Status:

Size: 1 Byte

Value	Parameter Description
0x00	Enter_Device_Under_Test_Mode command succeeded.
0x01-0xFF	Enter_Device_Under_Test_Mode command failed. See Table 6.1 on page 745 for list of Error Codes.

Event(s) generated (unless masked away):

When the Enter_Device_Under_Test_Mode command has completed, a Command Complete event will be generated.

5 EVENTS

5.1 EVENT

In addition to the events listed below, event code 0xFF is reserved for the event code used for vendor-specific debug events, and event code 0xFE is reserved for Bluetooth Logo Testing.

Event	Event Summary Description
Inquiry Complete event	The Inquiry Complete event indicates that the Inquiry is finished.
Inquiry Result event	The Inquiry Result event indicates that a Bluetooth device or multiple Bluetooth devices have responded so far during the current Inquiry process.
Connection Complete event	The Connection Complete event indicates to both of the Hosts forming the connection that a new connection has been established.
Connection Request event	The Connection Request event is used to indicate that a new incoming connection is trying to be established.
Disconnection Complete event	The Disconnection Complete event occurs when a connection has been terminated.
Authentication Complete event	The Authentication Complete event occurs when authentication has been completed for the specified connection.
Remote Name Request Complete event	The Remote Name Request Complete event is used to indicate a remote name request has been completed. The Remote_Name event parameter is a UTF-8 encoded string with up to 248 bytes in length.
Encryption Change event	The Encryption Change event is used to indicate that the change in the encryption has been completed for the Connection_Handle specified by the Connection_Handle event parameter.
Change Connection Link Key Complete event	The Change Connection Link Key Complete event is used to indicate that the change in the Link Key for the Connection_Handle specified by the Connection_Handle event parameter had been completed.
Master Link Key Complete event	The Master Link Key Complete event is used to indicate that the change in the temporary Link Key or in the semi-permanent link keys on the Bluetooth-master side has been completed.
Read Remote Supported Features Complete event	The Read Remote Supported Features Complete event is used to indicate the completion of the process of the Link Manager obtaining the supported features of the remote Bluetooth device specified by the Connection_Handle event parameter.

Table 5.1: List of Supported Events

Event	Event Summary Description
Read Remote Version Information Complete event	The Read Remote Version Information Complete event is used to indicate the completion of the process of the Link Manager obtaining the version information of the remote Bluetooth device specified by the Connection_Handle event parameter.
QoS Setup Complete event	The QoS Setup Complete event is used to indicate the completion of the process of the Link Manager setting up QoS with the remote Bluetooth device specified by the Connection_Handle event parameter.
Command Complete event	The Command Complete event is used by the Host Controller to pass the return status of a command and the other event parameters for each HCI Command.
Command Status event	The Command Status event is used to indicate that the command described by the Command_Opcode parameter has been received and the Host Controller is currently performing the task for this command.
Hardware Error event	The Error event is used to indicate some type of hardware failure for the Bluetooth device.
Flush Occurred event	The Flush Occurred event is used to indicate that, for the specified Connection Handle, the current user data to be transmitted has been removed.
Role Change event	The Role Change event is used to indicate that the current Bluetooth role related to the particular connection has been changed.
Number Of Completed Packets event	The Number Of Completed Packets event is used by the Host Controller to indicate to the Host how many HCI Data Packets have been completed for each Connection Handle since the previous Number Of Completed Packets event was sent.
Mode Change event	The Mode Change event is used to indicate when the device associated with the Connection Handle changes between Active, Hold, Sniff and Park mode.
Return Link Keys event	The Return Link Keys event is used to return stored link keys after a Read_Stored_Link_Key command is used.
PIN Code Request event	The PIN Code Request event is used to indicate that a PIN code is required to create a new link key for a connection.
Link Key Request event	The Link Key Request event is used to indicate that a Link Key is required for the connection with the device specified in BD_ADDR.
Link Key Notification event	The Link Key Notification event is used to indicate to the Host that a new Link Key has been created for the connection with the device specified in BD_ADDR.
Loopback Command event	The Loopback Command event is used to loop back most commands that the Host sends to the Host Controller.

Table 5.1: List of Supported Events

Event	Event Summary Description
Data Buffer Overflow event	The Data Buffer Overflow event is used to indicate that the Host Controller's data buffers have overflowed, because the Host has sent more packets than allowed.
Max Slots Change event	This event is used to notify the Host about the LMP_Max_Slots parameter when the value of this parameter changes.
Read Clock Offset Complete event	The Read Clock Offset Complete event is used to indicate the completion of the process of the LM obtaining the Clock offset information.
Connection Packet Type Changed event	The Connection Packet Type Changed event is used to indicate the completion of the process of the Link Manager changing the Packet Types used for the specified Connection_Handle.
QoS Violation event	The QoS Violation event is used to indicate the Link Manager is unable to provide the current QoS requirement for the Connection_Handle.
Page Scan Mode Change event	The Page Scan Mode Change event indicates that the connected remote Bluetooth device with the specified Connection_Handle has successfully changed the Page_Scan_Mode.
Page Scan Repetition Mode Change event	The Page Scan Repetition Mode Change event indicates that the connected remote Bluetooth device with the specified Connection_Handle has successfully changed the Page_Scan_Repetition_Mode (SR).

Table 5.1: List of Supported Events

THIS PAGE BLANK (USPTO)